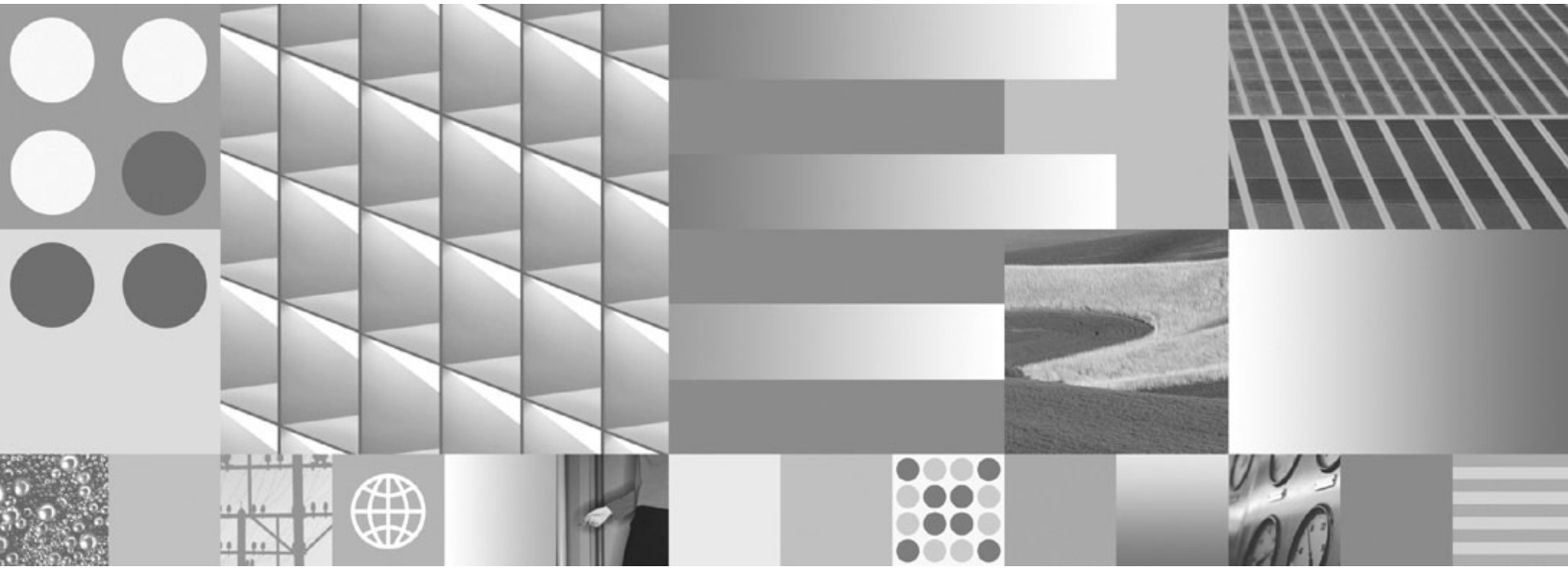
IBM DB2 9.7
for Linux, UNIX, and Windows

Version 9 Release 7

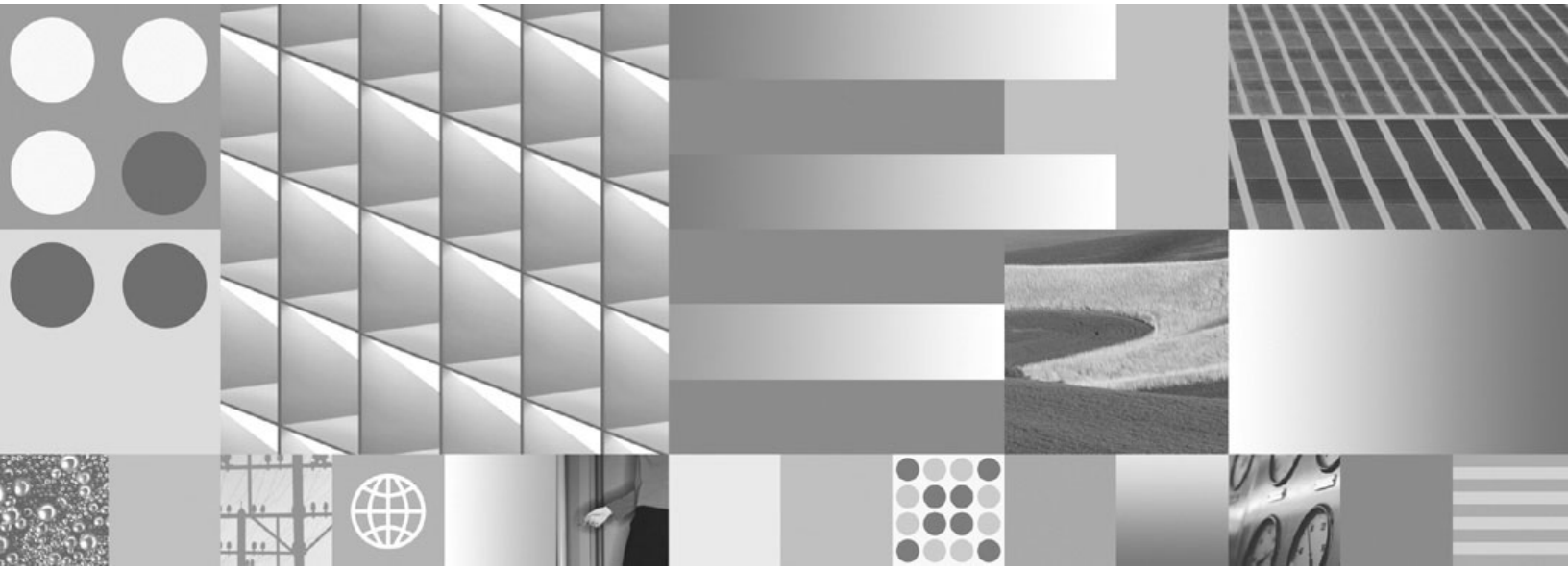


Administrative Routines and Views
Updated September, 2010

SC27-2436-02

IBM DB2 9.7
for Linux, UNIX, and Windows

Version 9 Release 7



Administrative Routines and Views
Updated September, 2010

SC27-2436-02

Note

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 1233.

Edition Notice

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. System-defined routines and views

The system-defined routines and views provide a primary, easy-to-use programmatic interface to administer and use DB2® through SQL. They encompass a collection of built-in views, table functions, procedures, and scalar functions for performing a variety of DB2 tasks. For example, system-defined routines are available for reorganizing a table, capturing and retrieving monitor data, or retrieving the application ID of the current connection.

These routines and views can be invoked from an SQL-based application, a DB2 command line, or a command script.

Best practices for calling system-defined routines and views in applications

To ensure your successful use of the system-defined routines and views, certain coding practices are recommended. These practices are especially important because at times the routines might change, as can happen from release to release as enhancements are made.

When you issue a query to retrieve information using a system-defined routine or view, do not use a statement of the form `SELECT * ...`. For example, do not issue the following query:

```
SELECT * FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,-1)) AS t
ORDER BY total_cpu_time DESC
```

Instead, name the result columns in the SELECT statement. This gives the application control over the number of result columns and the sequence in which they are returned. For example:

```
SELECT application_handle,
       uow_id,
       total_cpu_time,
       app_rqsts_completed_total,
       rqsts_completed_total
FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,-1)) AS t
ORDER BY total_cpu_time DESC
```

This prevents problems when the sequence and number of columns in the routines changes. It is possible the number of result columns that a routine returns might increase, and if, for example, you provide only five host variables when the routine returns six result columns, your application will break.

In addition, the type and size of output parameters or result columns of routines might change, for example a column might change from VARCHAR(8) to VARCHAR(128), or an INTEGER column might become a BIGINT column. If a variable you use is too small, the data you receive from the routine could be truncated.

To protect your application from such changes, for C applications, you can describe a prepared statement in order to determine which result columns are being returned and what are their types and sizes. For example, the following code snippet describes the query `SELECT application_handle, uow_id,total_cpu_time FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,-1)) AS t ORDER BY total_cpu_time DESC`:

```
strcpy(strStmt, "SELECT application_handle, uow_id,total_cpu_time
   FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,-1))
   AS t ORDER BY total_cpu_time DESC");
EXEC SQL PREPARE stmt FROM :strStmt;
EXEC SQL DESCRIBE stmt into :*pSqlda;
```

See the RowDatamemoryAlloc function in `samples/c/tbread.sqc` for more details regarding how to use the information returned in the SQLDA.

For Java™ and .Net applications, if data type and size is an issue, you can use metadata to determine which result columns are being returned and what are their types and sizes. For example:

```
ResultSet rs = pstmt.executeQuery();
ResultSetMetaData rsms = rs.getMetaData();
```

See the execPreparedQueryWithUnknownOutputColumn() method in `samples/java/jdbc/TbRead.java` for details regarding how to use the metadata of the result set.

Authorization for administrative views

For all administrative views in the SYSIBMADM schema, you need SELECT privilege on the view. This can be validated with the following query to check that your authorization ID, or a group or a role to which you belong, has SELECT privilege (that is, it meets the search criteria and is listed in the GRANTEE column):

```
SELECT GRANTEE, GRANTEETYPE
   FROM SYSCAT.TABAUTH
   WHERE TABSCHEMA = 'SYSIBMADM' AND TABNAME = '<view_name>' AND
   SELECTAUTH <> 'N'
```

where <view_name> is the name of the administrative view.

With the exception of SYSIBMADM.AUTHORIZATIONIDS, SYSIBMADM.OBJECTOWNERS, and SYSIBMADM.PRIVILEGES, you also need EXECUTE privilege on the underlying administrative table function. The underlying administrative table function is listed in the authorization section of the administrative view. This can be validated with the following query:

```
SELECT GRANTEE, GRANTEETYPE
   FROM SYSCAT.ROUTINEAUTH
   WHERE SCHEMA = 'SYSPROC' AND SPECIFICNAME = '<routine_name>' AND
   EXECUTEAUTH <> 'N'
```

where <routine_name> is the name of the underlying administrative table function as listed in the documentation.

Some administrative views require additional authorities beyond SELECT on the view and EXECUTE on the underlying administrative table function. Any additional authority required is documented in the reference information describing the view.

Administrative views versus table functions

DB2 Version 9.5 introduced administrative views that provide an easy-to-use application programming interface to DB2 administrative functions through SQL.

The administrative views fall into three categories:

- Views based on catalog views.
- Views based on table functions with no input parameters.
- Views based on table functions with one or more input parameters.

The administrative views are the preferred and only documented interfaces for the views based on catalog views and the views based on table functions with no input parameters because the table functions do not provide any additional information or performance benefits.

For administrative views based on table functions with one or more input parameters, both the administrative view and the table function can be used, each achieving a different goal:

- The ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V95 table function: The administrative view retrieves information for all tables in the database. This can have a significant performance impact for large databases. The performance impact can be reduced by using the table function and specifying a schema name, table name, or both as input.
- The PDLOGMSGS_LAST24HOURS administrative view and the PD_GET_LOG_MSGS table function: The administrative view, which retrieves notification log messages, provides quick access to data from the previous 24 hours, whereas the table function allows you to retrieve data from a specified period of time.
- All snapshot monitor administrative views and table functions (SNAP* administrative views, SNAP_GET_* table functions): The snapshot monitor administrative views provide access to data from each database partition. The table functions provide the option to choose between data from a single database partition or data aggregated across all database partitions.

Applications that use the table functions instead of the views might need to be changed because the table functions might change from release to release to enable new information to be returned. The new table function will have the same base name as the original function and will be suffixed with '_Vxx' for the version of the product in which it is added (for example, _V97). The administrative views will always be based on the most current version of the table functions, and therefore allow for more application portability. As the columns may vary from one release to the next, it is recommended that specific columns be selected from the administrative views, or that the result set be described if a SELECT * statement is used by an application.

Chapter 2. Supported system-defined SQL routines and views

The following tables summarize information about the supported administrative SQL routines and views.

Table 1. Activity monitor administrative SQL routines

Routine name	Schema	Description
AM_BASE_RPT_RECOMS table function	SYSPROC	This table function returns recommendations for activity reports used by the activity monitor.
AM_BASE_RPTS table function	SYSPROC	This table function returns activity reports used by the activity monitor.
AM_DROP_TASK procedure	SYSPROC	This procedure deletes a monitoring task.
AM_GET_LOCK_CHN_TB procedure	SYSPROC	This procedure returns application lock chain data in tabular format.
AM_GET_LOCK_CHNS procedure	SYSPROC	This procedure displays lock chains for a specified application using a formatted string.
AM_GET_LOCK_RPT procedure	SYSPROC	This procedure displays lock details for an application.
AM_GET_RPT procedure	SYSPROC	This procedure displays activity monitor data for a report.
AM_SAVE_TASK procedure	SYSPROC	This procedure creates or modifies a monitoring task.

Table 2. ADMIN_CMD stored procedure and associated administrative SQL routines

Routine name	Schema	Description
ADMIN_CMD procedure	SYSPROC	This procedure allows the administrator to execute administrative commands (including DB2 command line processor (CLP) commands) by running ADMIN_CMD through a CALL statement.
ADMIN_EST_INLINE_LENGTH function	SYSIBM	This function returns an estimate of the inline length that is required to inline the data stored in an XML column, BLOB column, CLOB column, or DBCLOB column.
ADMIN_GET_DBP_MEM_USAGE table function	SYSPROC	This table function gets the total memory consumption for a given instance.
ADMIN_GET_INDEX_COMPRESS_INFO	SYSPROC	This table function returns the potential index compression savings for uncompressed indexes or reports the index compression statistics from the catalogs.
ADMIN_GET_INDEX_INFO table function	SYSPROC	This table function returns index information not available in the catalog views.
ADMIN_GET_MSGS table function	SYSPROC	This table function is used to retrieve messages generated by data movement utilities that are executed through the ADMIN_CMD procedure.
ADMIN_IS_INLINED function	SYSIBM	This function retrieves state information about inline data for an XML column, BLOB column, CLOB column, or DBCLOB column.
ADMIN_REMOVE_MSGS procedure	SYSPROC	This procedure is used to clean up messages generated by data movement utilities that are executed through the ADMIN_CMD procedure.
ADMIN_REVALIDATE_DB_OBJECTS procedure	SYSPROC	This procedure revalidates invalid database objects.
ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This view and table function return compression information for tables, materialized query tables (MQT) and hierarchy tables.
ADMINTABINFO and ADMIN_GET_TAB_INFO_V97	SYSIBMADM (administrative view), SYSPROC (table function)	This view and table function return size and state information for tables, materialized query tables (MQT) and hierarchy tables.
ADMINTEMPCOLUMNS view and ADMIN_GET_TEMP_COLUMNS table function	SYSIBMADM (administrative view), SYSPROC (table function)	This view and table function retrieve column attribute information for created temporary tables and declared temporary tables
ADMINTEMPTABLES view and ADMIN_GET_TEMP_TABLES table function	SYSIBMADM (administrative view), SYSPROC (table function)	This view and table function retrieve table attribute and instantiation time information for instances of created temporary tables and declared temporary tables.

Table 3. Administrative task scheduler routines and views

Routine or view name	Schema	Description
ADMIN_TASK_ADD	SYSPROC	This procedure schedules an administrative task.
ADMIN_TASK_LIST	SYSTOOLS	This administrative view retrieves information about each task defined in the scheduler.
ADMIN_TASK_REMOVE	SYSPROC	This procedure removes scheduled tasks or task status records.
ADMIN_TASK_STATUS	SYSTOOLS	This administrative view retrieves information about the status of each task.
ADMIN_TASK_UPDATE	SYSPROC	This procedure updates an existing task

Table 4. Audit routines and procedures

Routine or view name	Schema	Description
AUDIT_ARCHIVE procedure and table function	SYSPROC	This procedure and table function archives the current audit log.
AUDIT_DELIM_EXTRACT procedure	SYSPROC	This procedure extracts data from the binary archived logs and loads it into delimited files.
AUDIT_LIST_LOGS table function	SYSPROC	This table function returns a list of the archived audit logs at the specified path, for the current database.

Table 5. Automatic Maintenance administrative SQL routines and views

Routine or view name	Schema	Description
AUTOMAINT_GET_POLICY procedure	SYSPROC	This procedure gets the current automatic maintenance settings for the database.
AUTOMAINT_GET_POLICYFILE procedure	SYSPROC	This procedure gets the current automatic maintenance settings for the database.
AUTOMAINT_SET_POLICY procedure	SYSPROC	This procedure sets the automatic maintenance policy settings for the currently connected database.
AUTOMAINT_SET_POLICYFILE procedure	SYSPROC	This procedure sets the automatic maintenance settings for the currently connected database.

Table 6. Common SQL API stored procedures

Routine or view name	Schema	Description
CANCEL_WORK procedure	SYSPROC	This procedure cancels a specified activity. If no unique activity ID is specified, cancels all activity for a connected application, and forces the application off of the system.
GET_CONFIG procedure	SYSPROC	This procedure retrieves data server configuration data, including `nodes.cfg` file data, database manager configuration data, database configuration data, and registry settings from all database partitions.

Table 6. Common SQL API stored procedures (continued)

Routine or view name	Schema	Description
GET_MESSAGE procedure	SYSPROC	This procedure retrieves the short message text, long message text, and SQLSTATE for an SQLCODE.
GET_SYSTEM_INFO procedure	SYSPROC	This procedure retrieves information about the data server, including information about the system, the current instance, installed DB2 database products, environment variables, available CPUs, and other system information.
SET_CONFIG procedure	SYSPROC	This procedure updates the configuration parameters retrieved by the GET_CONFIG procedure.

Table 7. Configuration administrative SQL routines and views

Routine or view name	Schema	Description
DB_PARTITIONS table function	SYSPROC	This table function returns the contents of the `db2nodes.cfg` file in table form.
DBCFG administrative view	SYSIBMADM	This administrative view returns database configuration information.
DBMCFG administrative view	SYSIBMADM	This administrative view returns database manager configuration information.
REG_VARIABLES administrative view	SYSIBMADM	This administrative view returns the DB2 registry settings from all database partitions.

Table 8. Environment administrative views

View name	Schema	Description
ENV_FEATURE_INFO administrative view	SYSPROC	This administrative view returns information about all available features for which a license is required.
ENV_INST_INFO administrative view	SYSIBMADM	This administrative view returns information about the current instance.
ENV_PROD_INFO administrative view	SYSIBMADM	This administrative view returns information about installed DB2 database products.
ENV_SYS_INFO administrative view	SYSIBMADM	This administrative view returns information about the system.
ENV_SYS_RESOURCES administrative view	SYSIBMADM	This administrative view returns operating system, CPU, memory and other information related to the system

Table 9. Health snapshot administrative SQL routines

Routine name	Schema	Description
HEALTH_CONT_HI table function	SYSPROC	This table function returns a table with health indicator information for containers from a health snapshot of a database.

Table 9. Health snapshot administrative SQL routines (continued)

Routine name	Schema	Description
HEALTH_CONT_HI_HIS table function	SYSPROC	This table function returns a table with health indicator history information for containers from a health snapshot of a database.
HEALTH_CONT_INFO table function	SYSPROC	This table function returns a table with rolled-up alert state information for containers from a health snapshot of a database.
HEALTH_DB_HI table function	SYSPROC	This table function returns a table with health indicator information from a health snapshot of a database.
HEALTH_DB_HI_HIS table function	SYSPROC	This table function returns a table with health indicator history information from a health snapshot of a database.
HEALTH_DB_HIC table function	SYSPROC	This table function returns collection health indicator information from a health snapshot of a database.
HEALTH_DB_HIC_HIS table function	SYSPROC	This table function returns collection health indicator history information from a health snapshot of a database.
HEALTH_DB_INFO table function	SYSPROC	This table function returns a table with rolled-up alert state information from a health snapshot of one or all databases.
HEALTH_DBM_HI table function	SYSPROC	This table function returns a table with health indicator information from a health snapshot of the DB2 database manager.
HEALTH_DBM_HI_HIS table function	SYSPROC	This table function returns a table with health indicator history information from a health snapshot of the DB2 database manager.
HEALTH_DBM_INFO table function	SYSPROC	This table function returns a table with rolled-up alert state information from a health snapshot of the DB2 database manager.
HEALTH_GET_ALERT_ACTION_CFG table function	SYSPROC	This table function returns health alert action configuration settings for objects of various types (dbm, database, table space, and table space containers) and for various configuration levels (install default, instance, global, and object).
HEALTH_GET_ALERT_CFG table function	SYSPROC	This table function returns health alert configuration settings for objects of various types (dbm, database, table space, table space containers) and for various configuration levels (install default, global, and object).
HEALTH_GET_IND_DEFINITION table function	SYSPROC	This table function returns the health indicator definition.
HEALTH_HI_REC procedure	SYSPROC	This procedure retrieves a set of recommendations that address a health indicator in alert state on a particular DB2 object.

Table 9. Health snapshot administrative SQL routines (continued)

Routine name	Schema	Description
HEALTH_TBS_HI table function	SYSPROC	This table function returns a table with health indicator information for table spaces from a health snapshot of a database.
HEALTH_TBS_HI_HIS table function	SYSPROC	This table function returns a table with health indicator history information for table spaces from a health snapshot of a database.
HEALTH_TBS_INFO table function	SYSPROC	This table function returns a table with rolled-up alert state information for table spaces from a health snapshot of a database.

Table 10. Monitor SQL routines

Routine name	Schema	Description
EVMON_FORMAT_UE_TO_TABLES procedure	SYSPROC	This procedure retrieves data stored in an unformatted event table and moves the XML document into a set of relational tables.
EVMON_FORMAT_UE_TO_TABLES table function	SYSPROC	This table function extracts binary events from an unformatted event table and formats them into an XML document.
MON_GET_ACTIVITY_DETAILS	SYSPROC	This table function returns details about an activity, including general activity information and a set of metrics for the activity.
MON_GET_BUFFERPOOL table function	SYSPROC	This table function returns monitor metrics for one or more buffer pools.
MON_GET_CONNECTION table function	SYSPROC	This table function returns metrics for one or more connections.
MON_GET_CONNECTION_DETAILS table function	SYSPROC	This table function returns detailed metrics for one or more connections.
MON_GET_CONTAINER table function	SYSPROC	This table function returns monitor metrics for one or more table space containers.
MON_GET_EXTENT_MOVEMENT_STATUS table function	SYSPROC	This table function returns the status of the extent movement operation.
MON_GET_INDEX table function	SYSPROC	This table function returns metrics for one or more indexes.
MON_GET_PKG_CACHE_STMT table function	SYSPROC	This table function returns a point-in-time view of both static and dynamic SQL statements in the database package cache.
MON_GET_SERVICE_SUBCLASS table function	SYSPROC	This table function returns metrics for one or more service subclasses.
MON_GET_SERVICE_SUBCLASS_DETAILS table function	SYSPROC	This table function returns detailed metrics for one or more service subclasses.
MON_GET_TABLE table function	SYSPROC	This table function returns monitor metrics for one or more tables.

Table 10. Monitor SQL routines (continued)

Routine name	Schema	Description
MON_GET_TABLESPACE table function	SYSPROC	This table function returns monitor metrics for one or more table spaces.
MON_GET_UNIT_OF_WORK table function	SYSPROC	This table function returns metrics for one or more units of work.
MON_GET_UNIT_OF_WORK_DETAILS table function	SYSPROC	This table function returns detailed metrics for one or more units of work.
MON_GET_WORKLOAD table function	SYSPROC	This table function returns metrics for one or more workloads.
MON_GET_WORKLOAD_DETAILS table function	SYSPROC	This table function returns detailed metrics for one or more workloads.

Table 11. MQSeries administrative SQL routines

Routine name	Schema	Description
MQPUBLISH scalar function	DB2MQ, DB2MQ1C	This scalar function publishes data to an MQSeries location.
MQREAD scalar function	DB2MQ, DB2MQ1C	This scalar function returns a message from an MQSeries location.
MQREADALL table function	DB2MQ, DB2MQ1C	This table function returns a table with messages and message metadata from an MQSeries location.
MQREADALLCLOB table function	DB2MQ	This table function returns a table containing messages and message metadata from a specified MQSeries location.
MQREADCLOB scalar function	DB2MQ	This scalar function returns a message from a specified MQSeries location.
MQRECEIVE scalar function	DB2MQ, DB2MQ1C	This scalar function returns a message from an MQSeries location and removes the message from the associated queue.
MQRECEIVEALL table function	DB2MQ, DB2MQ1C	This table function returns a table containing the messages and message metadata from an MQSeries location and removes the messages from the associated queue.
MQRECEIVEALLCLOB table function	DB2MQ	This table function returns a table containing messages and message metadata from a specified MQSeries location.
MQRECEIVECLOB scalar function	DB2MQ	This scalar function returns a message from a specified MQSeries location.
MQSEND scalar function	DB2MQ, DB2MQ1C	This scalar function sends data to an MQSeries location.
MQSUBSCRIBE scalar function	DB2MQ, DB2MQ1C	This scalar function subscribes to MQSeries messages published on a specific topic.
MQUNSUBSCRIBE scalar function	DB2MQ, DB2MQ1C	This scalar function unsubscribes from MQSeries messages published on a specific topic.

Table 12. Security administrative SQL routines and views:

Routine or view name	Schema	Description
AUTH_LIST_AUTHORITIES_FOR_AUTHID table function	SYSPROC	This table function returns all authorities held by the authorization ID, either found in the database configuration file or granted to an authorization ID directly or indirectly through a group or a role.
AUTH_LIST_GROUPS_FOR_AUTHID table function	SYSPROC	This table function returns the list of groups of which the given authorization ID is a member.
AUTH_LIST_ROLES_FOR_AUTHID function	SYSPROC	This function returns the list of roles in which the given authorization ID is a member.
AUTHORIZATIONIDS administrative view	SYSIBMADM	This administrative view contains a list of authorization IDs that have been granted privileges or authorities, along with their types, for the currently connected database.
OBJECTOWNERS administrative view	SYSIBMADM	This administrative view contains all object ownership information for the currently connected database.
PRIVILEGES administrative view	SYSIBMADM	This administrative view contains all explicit privileges for the currently connected database.

Table 13. Snapshot administrative SQL routines and views

Routine or view name	Schema	Description
APPL_PERFORMANCE administrative view	SYSIBMADM	This administrative view displays information about the rate of rows selected versus rows read per application.
APPLICATIONS administrative view	SYSIBMADM	This administrative view returns information on connected database applications.
BP_HITRATIO administrative view	SYSIBMADM	This administrative view returns bufferpool hit ratios, including total, data, and index, in the database.
BP_READ_IO administrative view	SYSIBMADM	This administrative view returns bufferpool read performance information.
BP_WRITE_IO administrative view	SYSIBMADM	This administrative view returns bufferpool write performance information per bufferpool.
CONTAINER_UTILIZATION administrative view	SYSIBMADM	This administrative view returns information about table space containers and utilization rates.
LOCKS_HELD administrative view	SYSIBMADM	This administrative view returns information on current locks held.
LOCKWAITS administrative view	SYSIBMADM	This administrative view returns information on locks that are waiting to be granted.
LOG_UTILIZATION administrative view	SYSIBMADM	This administrative view returns information about log utilization for the currently connected database.

Table 13. Snapshot administrative SQL routines and views (continued)

Routine or view name	Schema	Description
LONG_RUNNING_SQL administrative view	SYSIBMADM	This administrative view returns the longest running SQL statements in the currently connected database.
QUERY_PREP_COST administrative view	SYSIBMADM	This administrative view returns a list of statements with information about the time required to prepare the statement.
SNAP_WRITE_FILE procedure	SYSPROC	This procedure writes system snapshot data to a file in the `tmp` subdirectory of the instance directory.
SNAPAGENT administrative view and SNAP_GET_AGENT table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about agents from an application snapshot, in particular, the agent logical data group.
SNAPAGENT_MEMORY_POOL administrative view and SNAP_GET_AGENT_MEMORY_POOL table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about memory usage at the agent level.
SNAPAPPL administrative view and SNAP_GET_APPL_V95 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about applications from an application snapshot, in particular, the appl logical data group.
SNAPAPPL_INFO administrative view and SNAP_GET_APPL_INFO_V95 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about applications from an application snapshot, in particular, the appl_info logical data group.
SNAPBP administrative view and SNAP_GET_BP_V95 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about buffer pools from a bufferpool snapshot, in particular, the bufferpool logical data group.
SNAPBP_PART administrative view and SNAP_GET_BP_PART table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about buffer pools from a bufferpool snapshot, in particular, the bufferpool_nodeinfo logical data group.
SNAPCONTAINER administrative view and SNAP_GET_CONTAINER_V91 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return table space snapshot information from the tablespace_container logical data group.
SNAPDB administrative view and SNAP_GET_DB_V95 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information from the database (dbase) and database storage (db_storage_group) logical groupings.
SNAPDB_MEMORY_POOL administrative view and SNAP_GET_DB_MEMORY_POOL table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about memory usage at the database level for UNIX® operating systems only.
SNAPDBM administrative view and SNAP_GET_DBM_V95 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return the snapshot monitor DB2 database manager (dbm) logical grouping information.

Table 13. Snapshot administrative SQL routines and views (continued)

Routine or view name	Schema	Description
SNAPDBM_MEMORY_POOL administrative view and SNAP_GET_DBM_MEMORY_POOL table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about memory usage at the database manager.
SNAPDETAILLOG administrative view and SNAP_GET_DETAILLOG_V91 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information from the detail_log logical data group.
SNAPDYN_SQL administrative view and SNAP_GET_DYN_SQL_V95 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information from the dynsql logical data group.
SNAPFCM administrative view and SNAP_GET_FCM table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about the fast communication manager (FCM) from a database manager snapshot, in particular, the fcm logical data group.
SNAPFCM_PART administrative view and SNAP_GET_FCM_PART table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about the fast communication manager (FCM) from a database manager snapshot, in particular, the fcm_node logical data group.
SNAPHADR administrative view and SNAP_GET_HADR table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about high availability disaster recovery from a database snapshot, in particular, the hadr logical data group.
SNAPLOCK administrative view and SNAP_GET_LOCK table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information about locks, in particular, the lock logical data group.
SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information about lock waits, in particular, the lockwait logical data group.
SNAPSTMT administrative view and SNAP_GET_STMT table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about statements from an application snapshot.
SNAPSTORAGE_PATHS administrative view and SNAP_GET_STORAGE_PATHS table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return a list of automatic storage paths for the database including file system information for each storage path, specifically, from the db_storage_group logical data group
SNAPSUBSECTION administrative view and SNAP_GET_SUBSECTION table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about application subsections, namely the subsection logical monitor grouping.
SNAPSWITCHES administrative view and SNAP_GET_SWITCHES table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about the database snapshot switch state.

Table 13. Snapshot administrative SQL routines and views (continued)

Routine or view name	Schema	Description
SNAPTAB administrative view and SNAP_GET_TAB_V91 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information from the table logical data group.
SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return table reorganization information.
SNAPTBSP administrative view and SNAP_GET_TBSP_V91 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information from the table space logical data group.
SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V91 table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information from the tablespace_nodeinfo logical data group.
SNAPTBSP_QUIESCER administrative view and SNAP_GET_TBSP_QUIESCER table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about quiescers from a table space snapshot.
SNAPTBSP_RANGE administrative view and SNAP_GET_TBSP_RANGE table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information from a range snapshot.
SNAPUTIL administrative view and SNAP_GET_UTIL table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return snapshot information on utilities from the utility_info logical data group.
SNAPUTIL_PROGRESS administrative view and SNAP_GET_UTIL_PROGRESS table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return information about utility progress, in particular, the progress logical data group.
TBSP_UTILIZATION administrative view	SYSIBMADM	This administrative view returns table space configuration and utilization information.
TOP_DYNAMIC_SQL administrative view	SYSIBMADM	This administrative view returns the top dynamic SQL statements sortable by number of executions, average execution time, number of sorts, or sorts per statement.

Table 14. SQL procedures administrative SQL routines

Routine name	Schema	Description
ALTER_ROUTINE_PACKAGE procedure	SYSPROC	This procedure alters values for the package associated with a compiled SQL routine or a compiled trigger, without the need for rebinding.
GET_ROUTINE_OPTS scalar function	SYSPROC	This scalar function returns a character string value of the options that are to be used for the creation of SQL procedures in the current session.

Table 14. SQL procedures administrative SQL routines (continued)

Routine name	Schema	Description
GET_ROUTINE_SAR procedure	SYSFUN	This procedure returns the information necessary to install an identical routine on another database server running at least at the same level and operating system.
PUT_ROUTINE_SAR procedure	SYSFUN	This procedure passes the information necessary to create and define an SQL routine at the database server.
REBIND_ROUTINE_PACKAGE procedure	SYSPROC	This procedure rebinds the package associated with an SQL procedure.
SET_ROUTINE_OPTS procedure	SYSPROC	This procedure sets the options that are to be used for the creation of SQL procedures in the current session.

Table 15. Stepwise redistribute administrative SQL routines

Routine name	Schema	Description
ANALYZE_LOG_SPACE procedure	SYSPROC	This procedure returns log space analysis information.
GENERATE_DISTFILE procedure	SYSPROC	This procedure generates a data distribution file.
GET_SWRD_SETTINGS procedure	SYSPROC	This procedure returns redistribute information.
SET_SWRD_SETTINGS procedure	SYSPROC	This procedure creates or changes the redistribute registry.
STEPWISE_REDISTRIBUTE_DBPG procedure	SYSPROC	This procedure redistributes part of database partition group.

Table 16. Storage management tool administrative SQL routines

Routine name	Schema	Description
CAPTURE_STORAGEMGMT_INFO procedure	SYSPROC	This procedure returns storage-related information for a given root object.
CREATE_STORAGEMGMT_TABLES procedure	SYSPROC	This procedure creates storage management tables.
DROP_STORAGEMGMT_TABLES procedure	SYSPROC	This procedure drops all storage management tables.

Table 17. Text search administrative SQL routines

Routine name	Schema	Description
SYSTS_ADMIN_CMD stored procedure	SYSPROC	This procedure runs text search administrative commands using the SQL CALL statement.
SYSTS_ALTER procedure	SYSPROC	This procedure changes the update characteristics of an index.
SYSTS_CLEAR_COMMANDLOCKS procedure	SYSPROC	This procedure removes all command locks for a specific text search index or for all text search indexes in the database.

Table 17. Text search administrative SQL routines (continued)

Routine name	Schema	Description
SYSTS_CLEAR_EVENTS procedure	SYSPROC	This procedure deletes indexing events from an index's event table used for administration.
SYSTS_CREATE procedure	SYSPROC	This procedure creates a text search index for a text column which allows the column data to be searched using text search functions.
SYSTS_DISABLE procedure	SYSPROC	This procedure disables DB2 Text Search for the current database.
SYSTS_DROP procedure	SYSPROC	This procedure drops an existing text search index associated with any table column.
SYSTS_ENABLE procedure	SYSPROC	This procedure must be issued successfully before text search indexes on columns in tables within the database can be created.
SYSTS_UPDATE procedure	SYSPROC	This procedure updates the text search index to reflect the current contents of the text columns with which the index is associated.

Table 18. Workload management administrative SQL routines

Routine name	Schema	Description
WLM_CANCEL_ACTIVITY procedure	SYSPROC	This procedure cancels the given activity.
WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure	SYSPROC	This procedure sends information on the given activity to the activities event monitor.
WLM_COLLECT_STATS procedure	SYSPROC	This procedure sends statistics for service classes, workloads, work classes and threshold queues to the statistics event monitor and resets the in-memory copy of the statistics.
WLM_GET_QUEUE_STATS table function	SYSPROC	This table function returns basic statistic information for one or more threshold queues.
WLM_GET_SERVICE _CLASS_AGENTS_V97 table function	SYSPROC	This table function returns the list of agents on the given partition that are executing in the service class given by the SERVICE_SUPERCLASS_NAME and SERVICE_SUBCLASS_NAME or on behalf of the application given by the APPLICATION_HANDLE.
WLM_GET_SERVICE_CLASS_WORKLOAD _OCCURRENCES_V97 table function	SYSPROC	This table function returns the list of all workload occurrences executing in a given service class on a particular partition.
WLM_GET_SERVICE_SUBCLASS_STATS_V97 table function	SYSPROC	This table function returns basic statistics of one or more service subclasses.
WLM_GET_SERVICE_SUPERCLASS_STATS table function	SYSPROC	This table function returns basic statistics of one or more service superclasses.
WLM_GET_WORK_ACTION_SET_STATS table function	SYSPROC	This table function returns basic statistics for work classes in a work action set.

Table 18. Workload management administrative SQL routines (continued)

Routine name	Schema	Description
WLM_GET_WORKLOAD_OCCURRENCE _ACTIVITIES_V97 table function	SYSPROC	This table function returns the list of all activities that were submitted through the given application on the specified partition and have not yet completed.
WLM_GET_WORKLOAD_STATS_V97 table function	SYSPROC	This table function returns basic statistics for one or more workloads.
WLM_SET_CLIENT_INFO procedure	SYSPROC	This procedure sets client information associated with the current connection at the DB2 database server.

Table 19. Miscellaneous administrative SQL routines and views

Routine or view name	Schema	Description
ADMIN_COPY_SCHEMA procedure	SYSPROC	This procedure is used to copy a specific schema and all objects contained in it.
ADMIN_DROP_SCHEMA procedure	SYSPROC	This procedure is used to drop a specific schema and all objects contained in it.
ADMIN_MOVE_TABLE procedure	SYSPROC	This procedure moves data in an active table into a new table object with the same name, while the data remains online and available for access.
ADMIN_MOVE_TABLE_UTIL procedure	SYSPROC	This procedure alters the user definable values used by the ADMIN_MOVE_TABLE procedure.
ALTOBJ procedure	SYSPROC	This procedure alters an existing table using the input CREATE TABLE statement as the target table definition.
APPLICATION_ID scalar function	SYSFUN	This scalar function returns the application ID of the current connection.
COMPILATION_ENV table function	SYSPROC	This table function returns the elements of a compilation environment.
CONTACTGROUPS administrative view	SYSIBMADM	This administrative view returns the list of contact groups.
CONTACTS administrative view	SYSIBMADM	This administrative view returns the list of contacts defined on the database server.
DB_HISTORY administrative view	SYSIBMADM	This administrative view returns information from the history file that is associated with the currently connected database partition.
DBPATHS administrative view	SYSIBMADM	This administrative view returns the values for database paths required for tasks such as split mirror backups.
EXPLAIN_FORMAT_STATS scalar function	SYSPROC	This new scalar function is used to display formatted statistics information which is parsed and extracted from explain snapshot captured for a given query.
EXPLAIN_GET_MSGS table function	The schema is the same as the Explain table schema.	This table function queries the EXPLAIN_DIAGNOSTIC and EXPLAIN_DIAGNOSTIC_DATA Explain tables, and returns formatted messages.

Table 19. Miscellaneous administrative SQL routines and views (continued)

Routine or view name	Schema	Description
GET_DBSIZE_INFO procedure	SYSPROC	This procedure calculates the database size and maximum capacity.
NOTIFICATIONLIST administrative view	SYSIBMADM	This administrative view returns the list of contacts and contact groups that are notified about the health of an instance.
PD_GET_DIAG_HIST table function	SYSPROC	The table function returns log records, event records and notification records from a given facility.
PDLOGMSGS_LAST24HOURS administrative view and PD_GET_LOG_MSGS table function	SYSIBMADM (administrative view), SYSPROC (table function)	This administrative view and table function return problem determination log messages that were logged in the DB2 notification log. The information is intended for use by database and system administrators.
REORGCHK_IX_STATS procedure	SYSPROC	This procedure checks index statistics to determine whether or not there is a need for reorganization.
REORGCHK_TB_STATS procedure	SYSPROC	This procedure checks table statistics to determine whether or not there is a need for reorganization.
SQLERRM scalar function	SYSPROC	This scalar function has two versions. The first allows for full flexibility of message retrieval including using message tokens and language selection. The second is a simple interface which takes only an SQLCODE as an input parameter and returns the short message in English.
SYSINSTALLOBJECTS procedure	SYSPROC	This procedure creates or drops the database objects that are required for a specific tool.
MON_GET_FCM	SYSPROC	This table function returns metrics for the fast communication manager (FCM).
MON_GET_FCM_CONNECTION_LIST	SYSPROC	This table function returns monitor metrics for all the fast communication manager (FCM) connections on the specified member or members.

Chapter 3. Activity monitor routines

AM_BASE_RPT_RECOMS – Recommendations for activity reports

The AM_BASE_RPT_RECOMS table function returns recommendations for activity reports used by the activity monitor.

Syntax

The schema is SYSPROC.

Table function parameters

report-id
An input argument of type INTEGER that specifies a report ID. If the argument is null, recommendations for all available reports are returned.

client-locale
An input argument of type VARCHAR(33) that specifies a client language identifier. If the argument is null or an empty string, the default value is 'En_US' (English). If the message files for the specified locale are not available on the server, 'En_US' is used.

Authorization

EXECUTE privilege on the AM_BASE_RPT_RECOMS table function.

Examples

Example 1: Request recommendations (in English) for the activity monitor report with an ID of 1. Assume the default client language identifier 'En_US'.

```
SELECT *
   FROM TABLE(SYSPROC.AM_BASE_RPT_RECOMS(1, CAST(NULL AS VARCHAR(33))))
   AS RECOMS
```

Example 2: Request recommendations (in French) for the activity monitor report with an ID of 12.

```
SELECT *
   FROM TABLE(SYSPROC.AM_BASE_RPT_RECOMS(12, CAST('Fr_FR' AS VARCHAR(33))))
   AS RECOMS
```

Information returned

Table 20. Information returned by the AM_BASE_RPT_RECOMS table function

Column name	Data type	Description
REPORT_ID	INTEGER	The report ID.
RECOM_NAME	VARCHAR(256)	The name or short description of the recommendation.

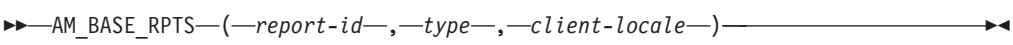
Table 20. Information returned by the AM_BASE_RPT_RECOMS table function (continued)

Column name	Data type	Description
RECOM_DESCRIPTION	CLOB(32K)	The detailed description of the recommendation.

AM_BASE_RPTS – Activity monitor reports

The AM_BASE_RPTS table function returns activity reports used by the activity monitor.

Syntax



The schema is SYSPROC.

Table function parameters

report-id
An input argument of type INTEGER that specifies a unique report ID. If the argument is null, reports with any report ID are returned.

type
An input argument of type CHAR(4) that specifies the report type. Valid values are:

APPL Application

STMT SQL statement

TRAN Transaction

CACH Dynamic SQL statement cache

Values can be specified in uppercase or lowercase characters. If the argument is null or an empty string, reports of any type are returned.

client-locale
An input argument of type VARCHAR(33) that specifies a client language identifier. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'En_US' is used.

Authorization

EXECUTE privilege on the AM_BASE_RPTS table function.

Examples

Example 1:

```
SELECT * FROM TABLE(SYSPROC.AM_BASE_RPTS(CAST(NULL AS INTEGER),
   CAST(NULL AS CHAR(4)), CAST(NULL AS VARCHAR(33)))) AS REPORTS
```

Example 2:

```
SELECT ID, NAME FROM TABLE(SYSPROC.AM_BASE_RPTS(
   CAST(NULL AS INTEGER), CAST('STMT' AS CHAR(4)), 'En_US'))
   AS REPORTS WHERE TYPE = 'STMT'
```

Information returned

Table 21. Information returned by the AM_BASE_RPTS table function

Column name	Data type	Description
ID	INTEGER	The unique report ID.
TYPE	CHAR(4)	The report type. Valid values are: APPL, STMT, TRAN, CACH.
NAME	VARCHAR(256)	The name or short description of the report.
DESCRIPTION	VARCHAR(16384)	The detailed description of the report.
SWITCHES	VARCHAR(100)	The monitor switches required for this report.

AM_DROP_TASK – Delete a monitoring task

The AM_DROP_TASK procedure deletes a monitoring task. It does not return any data.

Syntax

The schema is SYSPROC.

Procedure parameter

task-id
An input argument of type INTEGER that specifies a unique monitoring task ID.

Authorization

EXECUTE privilege on the AM_DROP_TASK procedure.

Example

Drop the monitoring task with ID 5.

```
CALL SYSPROC.AM_DROP_TASK(5)
```

AM_GET_LOCK_CHN_TB – Retrieve application lock chain data in a tabular format

The AM_GET_LOCK_CHN_TB procedure returns application lock chain data in tabular format. A lock chain consists of all the applications that the current application is holding up or waiting for, either directly or indirectly.

Syntax

The schema is SYSPROC.

Procedure parameters

agent-id
An input argument of type BIGINT that specifies the agent ID of the application for which lock chain data is to be retrieved.

Authorization

- SYSMON authority
- EXECUTE privilege on the AM_GET_LOCK_CHN_TB procedure.

Example

Retrieve lock chain information for agent ID 68.

```
CALL SYSPROC.AM_GET_LOCK_CHN_TB(68)
```

Information returned

The procedure returns a table as shown below. Each row of the table represents a lock-wait relationship. The result set also contains a row for each holding-only application; in this case, the HOLDING_AGENT_ID column is null, and the other four columns are for the holding-only application.

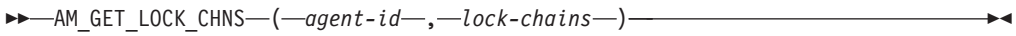
Table 22. Information returned by the AM_GET_LOCK_CHN_TB procedure

Column name	Data Type	Description
HOLDING_AGENT_ID	BIGINT	The agent ID of the application holding the lock.
AGENT_ID	BIGINT	The agent ID of the application waiting for the lock.
APPL_NAME	VARCHAR(255)	The name of the application waiting for the lock.
AUTH_ID	VARCHAR(128)	The authorization ID of the application waiting for the lock.
APPL_ID	VARCHAR(64)	The application ID of the application waiting for the lock.

AM_GET_LOCK_CHNS – Retrieve lock chain information for a specific application

The AM_GET_LOCK_CHNS procedure returns lock chains for the specified application as a formatted string. A lock chain consists of all the applications that the current application is holding up or waiting for, either directly or indirectly.

Syntax



The schema is SYSPROC.

Procedure parameters

agent-id
An input argument of type BIGINT that specifies the agent ID of the application whose lock chains are to be displayed.

lock-chains
An output argument of type CLOB(2M) that shows all the lock chains for the specified application.

Authorization

- SYSMON authority
- EXECUTE privilege on the AM_GET_LOCK_CHNS procedure.

Example

```
CALL SYSPROC.AM_GET_LOCK_CHNS(17,?)

  Value of output parameters
  --------------------------
  Parameter Name  : LOCK_CHAINS
  Parameter Value : >db2bp.exe (Agent ID: 17) (Auth ID: AMUSERC )

<db2bp.exe (Agent ID: 17) (Auth ID: AMUSERC )
    <db2bp.exe (Agent ID: 18) (Auth ID: AMUSERB )
        <db2bp.exe (Agent ID: 16) (Auth ID: AMUSERA )

  Return Status = 0
```

AM_GET_LOCK_RPT – Retrieve application lock details

The AM_GET_LOCK_RPT procedure returns lock details for an application in three output result sets.

Syntax

The schema is SYSPROC.

Procedure parameter

agent-id
An input argument of type BIGINT that specifies the agent ID of the application whose lock details are to be returned.

Authorization

- SYSMON authority
- EXECUTE privilege on the AM_GET_LOCK_RPT procedure.

Example

```
CALL SYSPROC.AM_GET_LOCK_RPT(68)
```

Usage note

The DFT_MON_LOCK monitor switch must be turned on for this procedure to return any information.

Information returned

The procedure returns three result sets: one for application general information; one for locks that the application holds; and one for locks that the application is waiting for.

Table 23. General application information returned by the AM_GET_LOCK_RPT procedure

Column name	Data Type	Description
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
APPL_NAME	VARCHAR(256)	appl_name - Application name
AUTH_ID	VARCHAR(128)	auth_id - Authorization ID
APPL_ID	VARCHAR(128)	appl_id - Application ID
APPL_STATUS	VARCHAR(22)	appl_status - Application status. This interface returns a text identifier based on the defines in sqlmon.h, and is one of: • BACKUP • COMMIT_ACT • COMP • CONNECTED • CONNECTPEND • CREATE_DB • DECOUPLED • DISCONNECTPEND • INTR • IOERROR_WAIT • LOAD • LOCKWAIT • QUIESCE_TABLESPACE • RECOMP • REMOTE_RQST • RESTART • RESTORE • ROLLBACK_ACT • ROLLBACK_TO_SAVEPOINT • TEND • THABRT • THCOMT • TPREP • UNLOAD • UOWEXEC • UOWWAIT • WAITFOR_REMOTE
COORD_PARTITION_NUM	SMALLINT	coord_node - Coordinating node
SEQUENCE_NO	VARCHAR(4)	sequence_no - Sequence number
CLIENT_PRDID	VARCHAR(128)	client_prdid - Client product/version ID
CLIENT_PID	BIGINT	client_pid - Client process ID

Table 23. General application information returned by the AM_GET_LOCK_RPT procedure (continued)

Column name	Data Type	Description
CLIENT_PLATFORM	VARCHAR(12)	client_platform - Client operating platform. This interface returns a text identifier based on the defines in sqlmon.h, • AIX® • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS • WINDOWS95

Table 23. General application information returned by the AM_GET_LOCK_RPT procedure (continued)

Column name	Data Type	Description
CLIENT_PROTOCOL	VARCHAR(10)	client_protocol - Client communication protocol. This interface returns a text identifier based on the defines in sqlmon.h, • CPIC • LOCAL • NETBIOS • NPIPE • TCPIP (for DB2 Universal Database™, or DB2 UDB) • TCPIP4 • TCPIP6
CLIENT_NNAME	VARCHAR(128)	The client_nname monitor element is deprecated. The value returned is not a valid value.
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAIT_START_TIME	TIMESTAMP	lock_wait_start_time - Lock wait start timestamp
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected

Table 24. Locks held information returned by the AM_GET_LOCK_RPT procedure

Column name	Data Type	Description
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name

Table 24. Locks held information returned by the AM_GET_LOCK_RPT procedure (continued)

Column name	Data Type	Description
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z

Table 24. Locks held information returned by the AM_GET_LOCK_RPT procedure (continued)

Column name	Data Type	Description
LOCK_STATUS	VARCHAR(10)	lock_status - Lock status. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • CONV • GRNT
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

Table 25. Locks wait information returned by the AM_GET_LOCK_RPT procedure

Column name	Data Type	Description
AGENT_ID_HOLDING_LK	BIGINT	agent_id_holding_lock - Agent ID holding lock
APPL_ID_HOLDING_LK	VARCHAR(128)	appl_id_holding_lk - Application ID holding lock
LOCK_WAIT_START_TIME	TIMESTAMP	lock_wait_start_time - Lock wait start timestamp
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name

Table 25. Locks wait information returned by the AM_GET_LOCK_RPT procedure (continued)

Column name	Data Type	Description
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z

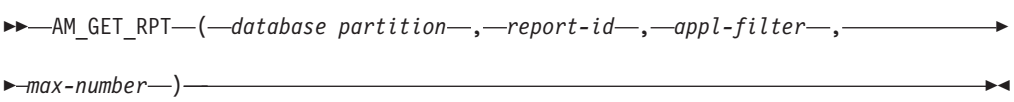
Table 25. Locks wait information returned by the AM_GET_LOCK_RPT procedure (continued)

Column name	Data Type	Description
LOCK_MODE_REQUESTED	VARCHAR(10)	lock_mode_requested - Lock mode requested. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation

AM_GET_RPT – Retrieve activity monitor data

The AM_GET_RPT procedure returns activity monitor data for a report.

Syntax



The schema is SYSPROC.

Procedure parameters

database partition
An input argument of type INTEGER that specifies a database partition number. Valid values are -2 (denoting all database partitions) and the database partition number of any existing database partition.

report-id
An input argument of type INTEGER that specifies a unique report ID.

appl-filter
An input argument of type CLOB(32K) that specifies an application filter. An application filter is a search condition involving any or all of the three columns AGENT_ID, APPL_NAME, and AUTH_ID, where AGENT_ID and AUTH_ID are integers, and APPL_NAME is a character string. If the argument is null or an empty string, no filtering is performed.

max-number
An input argument of type INTEGER that specifies the maximum number of

applications, statements, or transactions that are to be displayed. If the argument is null, all applications, statements, and transactions will be displayed.

Authorization

- SYSMON authority
- EXECUTE privilege on the AM_GET_RPT procedure.

Example

```
CALL SYSPROC.AM_GET_RPT(-2, 18,
   CAST('AGENT_ID=29 AND AUTH_ID <> ''dbuser'' AND APPL_NAME LIKE ''db2%'''
       AS CLOB(32K)), 100)
```

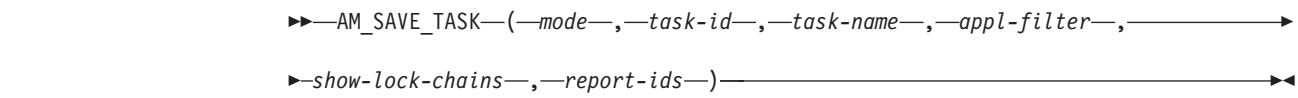
Usage note

The result set returned is different for each report id. This procedure is intended to support the Activity Monitor graphical tool. To build reports that can be parsed, snapshot administrative SQL routines and views should be used instead.

AM_SAVE_TASK – Create or modify a monitoring task

The AM_SAVE_TASK procedure creates or modifies a monitoring task.

Syntax



The schema is SYSPROC.

Procedure parameters

mode
An input argument of type CHAR(1) that specifies whether to create a new monitoring task ('C') or to modify an existing monitoring task ('M').

task-id
An input argument of type INTEGER that specifies a unique monitoring task ID. When *mode* is 'C', any specified input for *task-id* is ignored. An ID for the new monitoring task will be generated by the procedure and returned in the output. When *mode* is 'M', specifies the ID of the monitoring task that is being modified.

task-name
An input argument of type VARCHAR(128) that specifies a name or short description for a monitoring task.

appl-filter
An input argument of type CLOB(32K) that specifies an application filter. An application filter is a search condition involving any or all of the three columns AGENT_ID, APPL_NAME, and AUTH_ID, where AGENT_ID and AUTH_ID are integers, and APPL_NAME is a character string. If the argument is null or an empty string, no filtering is performed.

show-lock-chains
An input argument of type CHAR(1) that specifies whether lock chains are to be shown. Valid values are 'Y' and 'N'. If the argument is null, lock chains are not to be shown.

report-ids
An input argument of type VARCHAR(3893) that specifies one or more report IDs separated by commas.

Authorization

EXECUTE privilege on the AM_SAVE_TASK procedure.

Example

Example:

```
CALL SYSPROC.AM_SAVE_TASK('M',11,'Task ABC',CAST (NULL AS CLOB(32K)),
   'N','1,2,4,8,9,12')
```

Chapter 4. ADMIN_CMD procedure and associated routines

ADMIN_CMD – Run administrative commands

The ADMIN_CMD procedure is used by applications to run administrative commands using the SQL CALL statement.

Syntax

The schema is SYSPROC.

Procedure parameter

command-string
An input argument of type CLOB (2M) that specifies a single command that is to be executed.

Authorization

EXECUTE privilege on the ADMIN_CMD procedure.

The procedure currently supports the following DB2 command line processor (CLP) commands:

- ADD CONTACT
- ADD CONTACTGROUP
- AUTOCONFIGURE
- BACKUP - online only
- DESCRIBE
- DROP CONTACT
- DROP CONTACTGROUP
- EXPORT
- FORCE APPLICATION
- IMPORT
- INITIALIZE TAPE
- LOAD
- PRUNE HISTORY/LOGFILE
- QUIESCE DATABASE
- QUIESCE TABLESPACES FOR TABLE
- REDISTRIBUTE
- REORG INDEXES/TABLE
- RESET ALERT CONFIGURATION
- RESET DATABASE CONFIGURATION
- RESET DATABASE MANAGER CONFIGURATION
- REWIND TAPE

- RUNSTATS
- SET TAPE POSITION
- UNQUIESCE DATABASE
- UPDATE ALERT CONFIGURATION
- UPDATE CONTACT
- UPDATE CONTACTGROUP
- UPDATE DATABASE CONFIGURATION
- UPDATE DATABASE MANAGER CONFIGURATION
- UPDATE HEALTH NOTIFICATION CONTACT LIST
- UPDATE HISTORY

Note: Some commands might have slightly different supported syntax when executed through the ADMIN_CMD procedure.

The procedure also supports the following commands which are not supported by the CLP:

- GET STMM TUNING DBPARTITIONNUM
- UPDATE STMM TUNING DBPARTITIONNUM

Usage notes

Retrieving command execution information:

- As the ADMIN_CMD procedure runs on the server, the utility messages are also created on the server. The MESSAGES ON SERVER option (refer to the specific command for further details) indicates that the message file is to be created on the server.
- Command execution status is returned in the SQLCA resulting from the CALL statement.
- If the execution of the administrative command is successful, and the command returns more than the execution status, the additional information is returned in the form of a result set (up to two result sets). For example, if the EXPORT command executes successfully, the returned result set contains information about the number of exported rows; however, if the RUNSTATS command executes successfully, no result set is returned. The result set information is documented with the corresponding command.
- If the execution of the administrative command is not successful, an SQL20397W warning message is returned by the ADMIN_CMD procedure along with a result set containing more details about the reason for the failure of the administrative command. Any application that uses the ADMIN_CMD procedure should check the SQLCODE returned by the procedure. If the SQLCODE is >= 0, the result set for the administrative command should be retrieved. The following table indicates what information might be returned depending on whether the MESSAGES ON SERVER option is used or not.

Table 26. SQLCODE and information returned by the ADMIN_CMD procedure

Administrative command execution status	MESSAGES ON SERVER option specified	MESSAGES ON SERVER option not specified
Successful	The SQLCODE returned is >= 0: Additional information (result sets) returned, if any.	The SQLCODE returned is >= 0: Additional information (result sets) returned, if any, but the MSG_RETRIEVAL and MSG_REMOVAL columns are NULL.
Failed	The SQLCODE returned 20397: Additional information (result sets) returned, but only the MSG_RETRIEVAL and MSG_REMOVAL columns are populated.	The SQLCODE returned is < 0: No additional information (result sets) is returned.

- The result sets can be retrieved from the CLP or from applications such as JDBC and DB2 CLI applications, but not from embedded C applications.
- Case-sensitive names and double-byte character set (DBCS) names must be enclosed inside a backward slash and double quotation delimiter, for example, \" MyTabLe \".

For all commands executed through the ADMIN_CMD, the user ID that established the connection to the database is used for authentication.

Any additional authority required, for example, for commands that need file system access on the database server, is documented in the reference information describing the command.

This procedure cannot be called from a user-defined function (SQLSTATE 38001) or a trigger.

ADD CONTACT command using the ADMIN_CMD procedure

Adds a contact to the contact list which can be either defined locally on the system or in a global list. Contacts are users to whom processes such as the Scheduler and Health Monitor send messages.

The setting of the Database Administration Server (DAS) **contact_host** configuration parameter determines whether the list is local or global.

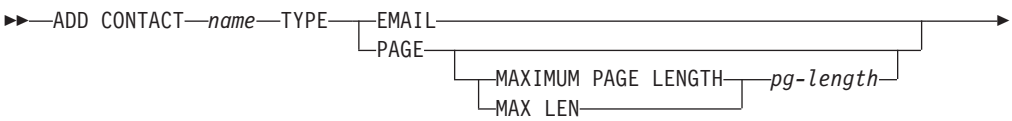
Authorization

None

Required connection

Database. The DAS must be running.

Command syntax



Command parameters

ADD CONTACT *name*
The name of the contact that will be added. By default the contact will be added in the local system, unless the DB2 administration server configuration parameter **contact_host** points to another system.

TYPE Method of contact, which must be one of the following two:

EMAIL
This contact wishes to be notified by e-mail at (**ADDRESS**).

PAGE This contact wishes to be notified by a page sent to **ADDRESS**.

MAXIMUM PAGE LENGTH *pg-length*
If the paging service has a message-length restriction, it is specified here in characters.

The notification system uses the SMTP protocol to send the notification to the mail server specified by the DB2 Administration Server configuration parameter **smtp_server**. It is the responsibility of the SMTP server to send the e-mail or call the pager.

ADDRESS *recipients-address*
The SMTP mailbox address of the recipient. For example, joe@somewhere.org. The **smtp_server** DAS configuration parameter must be set to the name of the SMTP server.

DESCRIPTION *contact description*
A textual description of the contact. This has a maximum length of 128 characters.

Example

Add a contact for user 'testuser' with e-mail address 'testuser@test.com'.

```
CALL SYSPROC.ADMIN_CMD
   ('ADD CONTACT testuser TYPE EMAIL ADDRESS testuser@test.com')
```

Usage notes

The DAS must have been created and be running.

Command execution status is returned in the SQLCA resulting from the CALL statement.

ADD CONTACTGROUP command using the ADMIN_CMD procedure

Adds a new contact group to the list of groups defined on the local system. A contact group is a list of users and groups to whom monitoring processes such as the Scheduler and Health Monitor can send messages.

The setting of the Database Administration Server (DAS) **contact_host** configuration parameter determines whether the list is local or global.

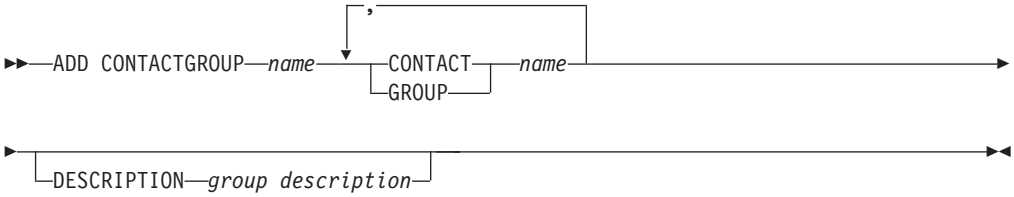
Authorization

None

Required connection

Database. The DAS must be running.

Command Syntax



Command Parameters

ADD CONTACTGROUP *name*
Name of the new contact group, which must be unique among the set of groups on the system.

CONTACT *name*
Name of the contact which is a member of the group. A contact can be defined with the ADD CONTACT command after it has been added to a group.

GROUP *name*
Name of the contact group of which this group is a member.

DESCRIPTION *group description*
Optional. A textual description of the contact group.

Example

Create a contact group named 'gname1' that contains two contacts: 'cname1' and 'cname2'.

```
CALL SYSPROC.ADMIN_CMD( 'add contactgroup gname1 contact cname1, contact cname2' )
```

Usage notes

The DAS must have been created and be running.

Command execution status is returned in the SQLCA resulting from the CALL statement.

AUTOCONFIGURE command using the ADMIN_CMD procedure

Calculates and displays initial values for the buffer pool size, database configuration and database manager configuration parameters, with the option of applying these recommended values.

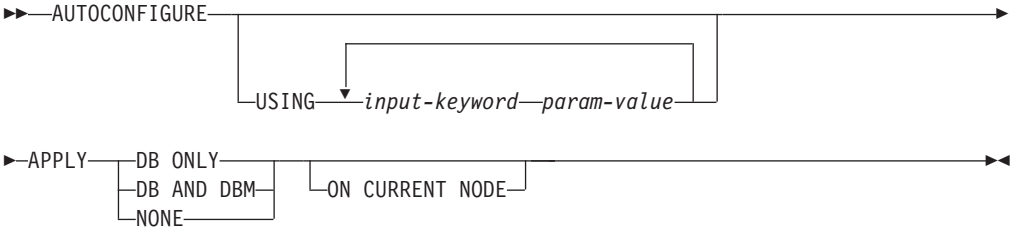
Authorization

SYSADM

Required connection

Database

Command syntax



Command parameters

USING *input-keyword param-value*

Table 27. Valid input keywords and parameter values

Keyword	Valid values	Default value	Explanation
mem_percent	1–100	25	Percentage of instance memory that is assigned to the database. However, if the CREATE DATABASE command invokes the configuration advisor and you do not specify a value for **mem_percent**, the percentage is calculated based on memory usage in the instance and the system up to a maximum of 25% of the instance memory.
workload_type	simple, mixed, complex	mixed	Simple workloads tend to be I/O intensive and mostly transactions, whereas complex workloads tend to be CPU intensive and mostly queries.
num_stmts	1–1 000 000	10	Number of statements per unit of work
tpm	1–200 000	60	Transactions per minute
admin_priority	performance, recovery, both	both	Optimize for better performance (more transactions per minute) or better recovery time
is_populated	yes, no	yes	Is the database populated with data?
num_local_apps	0–5 000	0	Number of connected local applications
num_remote_ apps	0–5 000	10	Number of connected remote applications

Table 27. Valid input keywords and parameter values (continued)

Keyword	Valid values	Default value	Explanation
isolation	RR, RS, CS, UR	RR	Maximum isolation level of applications connecting to this database (Repeatable Read, Read Stability, Cursor Stability, Uncommitted Read). It is only used to determine values of other configuration parameters. Nothing is set to restrict the applications to a particular isolation level and it is safe to use the default value.
bp_resizeable	yes, no	yes	Are buffer pools resizeable?

APPLY

DB ONLY
Displays the recommended values for the database configuration and the buffer pool settings based on the current database manager configuration. Applies the recommended changes to the database configuration and the buffer pool settings.

DB AND DBM
Displays and applies the recommended changes to the database manager configuration, the database configuration, and the buffer pool settings.

NONE
Displays the recommended changes, but does not apply them.

ON CURRENT NODE
In a partitioned database environment, the Configuration Advisor updates the database configuration on all nodes by default. Running with the **ON CURRENT NODE** option makes the advisor apply the recommended database configuration to the coordinator (connection) node only.

The buffer pool changes are always applied to the system catalogs. Thus, all nodes are affected. The **ON CURRENT NODE** option does not matter for buffer pool recommendations.

Example

Invoke autoconfigure on a database through the ADMIN_CMD stored procedure.

```
CALL SYSPROC.ADMIN_CMD( 'AUTOCONFIGURE APPLY NONE' )
```

The following is an example of the result set returned by the command.

```
LEVEL     NAME                 VALUE       RECOMMENDED_VALUE  DATATYPE
-----...- -----------------...- -------...- ------------------ --------...-
DBM       ASLHEAPSZ            15          15                 BIGINT
DBM       FCM_NUM_BUFFERS      512         512                BIGINT
...
DB        APP_CTL_HEAP_SZ      128         144                INTEGER
DB        APPGROUP_MEM_SZ      20000       14559              BIGINT
...
BP        IBMDEFAULTBP         1000        164182             BIGINT
```

Usage notes

- This command makes configuration recommendations for the currently connected database and assumes that the database is the only active database on

the instance. If you have not enabled the self tuning memory manager and you have more than one active database on the instance, specify a **mem_percent** value that reflects the database memory distribution. For example, if you have two active databases on the instance that should use 80% of the instance memory and should share the resources equally, specify 40% (80% divided by 2 databases) as the **mem_percent** value.

- If you have multiple instances on the same computer and the self tuning memory manager is not enabled, you should set a fixed value for **instance_memory** on each instance or specify a **mem_percent** value that reflects the database memory distribution. For example, if all active databases should use 80% of the computer memory and there are 4 instances each with one database, specify 20% (80% divided by 4 databases) as the **mem_percent** value.
- When explicitly invoking the Configuration Advisor with the AUTOCONFIGURE command, the setting of the **DB2_ENABLE_AUTOCONFIG_DEFAULT** registry variable will be ignored.
- Running the AUTOCONFIGURE command on a database will recommend enablement of the Self Tuning Memory Manager. However, if you run the AUTOCONFIGURE command on a database in an instance where **sheapthres** is not zero, sort memory tuning (**sortheap**) will not be enabled automatically. To enable sort memory tuning (**sortheap**), you must set **sheapthres** equal to zero using the UPDATE DATABASE MANAGER CONFIGURATION command. Note that changing the value of **sheapthres** may affect the sort memory usage in your previously existing databases.
- Command execution status is returned in the SQLCA resulting from the CALL statement.
- SQL executed in the ADMIN_CMD procedure on behalf of AUTOCONFIGURE is monitored by Query Patroller.
- The AUTOCONFIGURE command issues a COMMIT statement at the end if its execution. In the case of Type-2 connections this will cause the ADMIN_CMD procedure to return SQL30090N with reason code 2.

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the command returns additional information the following result set:

Table 28. Result set returned by the AUTOCONFIGURE command

Column name	Data type	Description
LEVEL	VARCHAR(3)	Level of parameter and is one of: • `BP` for buffer pool level • `DBM` for database manager level • `DB` for database level
NAME	VARCHAR(128)	• If LEVEL is `DB` or `DBM`, this contains the configuration parameter keyword. • If LEVEL is `BP`, this value contains the buffer pool name.

Table 28. Result set returned by the AUTOCONFIGURE command (continued)

Column name	Data type	Description
VALUE	VARCHAR(256)	• If LEVEL is DB or DBM, and the recommended values were applied, this column contains the value of the configuration parameter identified in the NAME column prior to applying the recommended value (that is, it contains the old value). If the change was not applied, this column contains the current on-disk (deferred value) of the identified configuration parameter. • If LEVEL is BP, and the recommended values were applied, this column contains the size (in pages) of the buffer pool identified in the NAME column prior to applying the recommended value (that is, it contains the old size). If the change was not applied, this column contains the current size (in pages) of the identified buffer pool.
RECOMMENDED_VALUE	VARCHAR(256)	• If LEVEL is DB or DBM, this column contains the recommended (or applied) value of the configuration parameter identified in the parameter column. • If type is BP, this column contains the recommended (or applied) size (in pages) of the buffer pool identified in the parameter column.
DATATYPE	VARCHAR(128)	Parameter data type.

BACKUP DATABASE command using the ADMIN_CMD procedure

Creates a backup copy of a database or a table space.

For information on the backup operations supported by DB2 database systems between different operating systems and hardware platforms, see "Backup and restore operations between different operating systems and hardware platforms".

Scope

In a partitioned database environment, if no database partitions are specified, this command affects only the database partition on which it is executed.

If the option to perform a partitioned backup is specified, the command can be called only on the catalog node. If the option specifies that all database partition

servers are to be backed up, it affects all database partition servers that are listed in the db2nodes.cfg file. Otherwise, it affects the database partition servers that are specified on the command.

Authorization

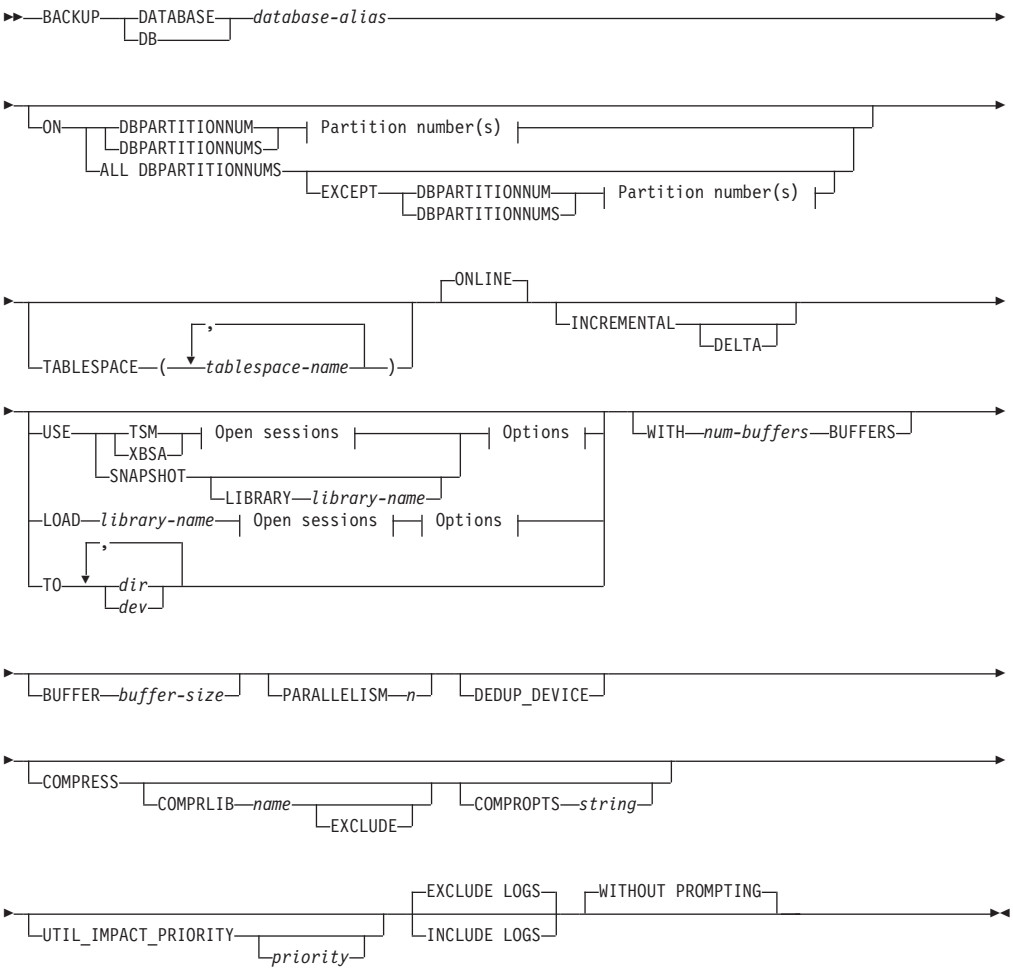
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT

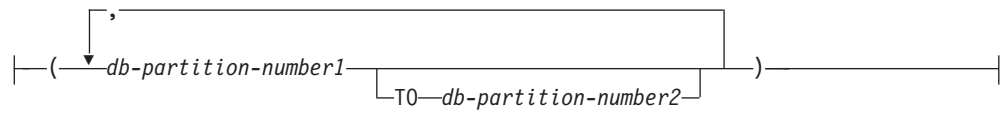
Required connection

Database. The existing database connection remains after the completion of the backup operation.

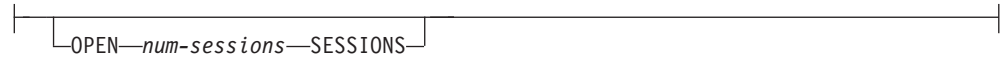
Command syntax



Partition number(s):



Open sessions:



Options:

Command parameters

DATABASE | DB *database-alias*
Specifies the alias of the database to back up. The alias must be a local database defined on the server and must be the database name that the user is currently connected to. If the database-alias is not the one the user is connected to, an SQL20322N error is returned.

ON Backup the database on a set of database partitions. This clause shall be specified only on the catalog partition.

DBPARTITIONNUM *db-partition-number1*
Specifies a database partition number in the database partition list.

DBPARTITIONNUMS *db-partition-number1* **TO** *db-partition-number2*
Specifies a range of database partition numbers, so that all partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

ALL DBPARTITIONNUMS
Specifies that the database is to be backed up on all partitions specified in the `db2nodes.cfg` file.

EXCEPT
Specifies that the database is to be backed up on all partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

DBPARTITIONNUM *db-partition-number1*
Specifies a database partition number in the database partition list.

DBPARTITIONNUMS *db-partition-number1* **TO** *db-partition-number2*
Specifies a range of database partition numbers, so that all partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

TABLESPACE *tablespace-name*
A list of names used to specify the table spaces to be backed up.

ONLINE

Specifies online backup. This is the only supported mode and is the default. The **ONLINE** clause does not need to be specified.

INCREMENTAL
Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DELTA
Specifies a non-cumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

USE

TSM Specifies that the backup is to use Tivoli® Storage Manager (TSM) output.

XBSA Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

SNAPSHOT
Specifies that a snapshot backup is to be taken.

You cannot use the **SNAPSHOT** parameter with any of the following parameters:

- **TABLESPACE**
- **INCREMENTAL**
- **WITH** *num-buffers* **BUFFERS**
- **BUFFER**
- **PARALLELISM**
- **COMPRESS**
- **UTIL_IMPACT_PRIORITY**
- **SESSIONS**

The default behavior for a snapshot backup is a FULL DATABASE OFFLINE backup of all paths that make up the database including all containers, local volume directory, database path (**DBPATH**), and primary log and mirror log paths (**INCLUDE LOGS** is the default for all snapshot backups unless **EXCLUDE LOGS** is explicitly stated).

LIBRARY *library-name*
Integrated into IBM® Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage® SAN Volume Controller
- IBM Enterprise Storage Server® Model 800
- IBM System Storage® DS6000™
- IBM System Storage DS8000®
- IBM System Storage N Series
- NetApp V-series

- NetApp FAS

If you have other storage hardware, and a DB2 ACS API driver for that storage hardware, you can use the **LIBRARY** parameter to specify the DB2 ACS API driver.

The value of the **LIBRARY** parameter is a fully-qualified library file name.

OPTIONS

"*options-string*"
Specifies options to be used for the backup operation. The string will be passed exactly as it was entered, without the double quotation marks.

@ *file-name*
Specifies that the options to be used for the backup operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library. The file must be a fully qualified file name.

You cannot use the **vendoropt** database configuration parameter to specify vendor-specific options for snapshot backup operations. You must use the **OPTIONS** parameter of the backup utilities instead.

OPEN *num-sessions* **SESSIONS**
The number of I/O sessions to be created between DB2 and TSM or another backup vendor product. This parameter has no effect when backing up to tape, disk, or other local device.

TO *dir* | *dev*
A list of directory or tape device names. The full path on which the directory resides must be specified. This target directory or device must exist on the database server.

In a partitioned database, the target directory or device must exist on all database partitions, and can optionally be a shared path. The directory or device name may be specified using a database partition expression. For more information about database partition expressions, see "Automatic storage databases".

This parameter can be repeated to specify the target directories and devices that the backup image will span. If more than one target is specified (target1, target2, and target3, for example), target1 will be opened first. The media header and special files (including the configuration file, table space table, and history file) are placed in target1. All remaining targets are opened, and are then used in parallel during the backup operation. Because there is no general tape support on Windows® operating systems, each type of tape device requires a unique device driver.

Use of tape devices or floppy disks might require prompts and user interaction, which will result in an error being returned.

If the tape system does not support the ability to uniquely reference a backup image, it is recommended that multiple backup copies of the same database not be kept on the same tape.

LOAD *library-name*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can

contain the full path. If the full path is not given, it will default to the path on which the user exit program resides.

WITH *num-buffers* **BUFFERS**
The number of buffers to be used. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. However, when creating a backup to multiple locations, a larger number of buffers can be used to improve performance.

BUFFER *buffer-size*
The size, in 4 KB pages, of the buffer used when building the backup image. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

If using tape with variable block size, reduce the buffer size to within the range that the tape device supports. Otherwise, the backup operation might succeed, but the resulting image might not be recoverable.

With most versions of Linux®, using DB2's default buffer size for backup operations to a SCSI tape device results in error SQL2025N, reason code 75. To prevent the overflow of Linux internal SCSI buffers, use this formula:

```
bufferpages <= ST_MAX_BUFFERS * ST_BUFFER_BLOCKS / 4
```

where *bufferpages* is the value you want to use with the **BUFFER** parameter, and `ST_MAX_BUFFERS` and `ST_BUFFER_BLOCKS` are defined in the Linux kernel under the `drivers/scsi` directory.

PARALLELISM *n*
Determines the number of table spaces which can be read in parallel by the backup utility. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value.

DEDUP_DEVICE
Optimizes the format of the backup images for target storage devices that support data deduplication. Available in Version 9.7 Fix Pack 3 and later fix packs.

UTIL_IMPACT_PRIORITY *priority*
Specifies that the backup will run in throttled mode, with the priority specified. Throttling allows you to regulate the performance impact of the backup operation. Priority can be any number between 1 and 100, with 1 representing the lowest priority, and 100 representing the highest priority. If the **UTIL_IMPACT_PRIORITY** keyword is specified with no priority, the backup will run with the default priority of 50. If **UTIL_IMPACT_PRIORITY** is not specified, the backup will run in unthrottled mode. An impact policy must be defined by setting the **util_impact_lim** configuration parameter for a backup to run in throttled mode.

COMPRESS
Indicates that the backup is to be compressed.

COMPRLIB *name*
Indicates the name of the library to be used to perform the compression (for example, `db2compr.dll` for Windows; `libdb2compr.so` for Linux or UNIX systems). The name must be a fully qualified path referring to a file on the server. If this

parameter is not specified, the default DB2 compression library will be used. If the specified library cannot be loaded, the backup will fail.

EXCLUDE
Indicates that the compression library will not be stored in the backup image.

COMPROPTS *string*
Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte reversal or code page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of string with the contents of this file and will pass this new value to the initialization routine instead. The maximum length for *string* is 1024 bytes.

EXCLUDE LOGS
Specifies that the backup image should not include any log files. When performing an offline backup operation, logs are excluded whether or not this option is specified, with the exception of snapshot backups.

INCLUDE LOGS
Specifies that the backup image should include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup, with the exception of snapshot backups where this option is the default unless explicitly told to exclude.

WITHOUT PROMPTING

Specifies that the backup will run unattended, and that any actions which normally require user intervention will return an error message. This is the default.

Examples

The following is a sample weekly incremental backup strategy for a recoverable database. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

```
(Sun) CALL SYSPROC.ADMIN_CMD('backup db sample online use tsm')
(Mon) CALL SYSPROC.ADMIN_CMD
      ('backup db sample online incremental delta use tsm')
(Tue) CALL SYSPROC.ADMIN_CMD
      ('backup db sample online incremental delta use tsm')
(Wed) CALL SYSPROC.ADMIN_CMD
      ('backup db sample online incremental use tsm')
(Thu) CALL SYSPROC.ADMIN_CMD
      ('backup db sample online incremental delta use tsm')
(Fri) CALL SYSPROC.ADMIN_CMD
      ('backup db sample online incremental delta use tsm')
(Sat) CALL SYSPROC.ADMIN_CMD
      ('backup db sample online incremental use tsm')
```

Usage notes

The data in a backup cannot be protected by the database server. Make sure that backups are properly safeguarded, particularly if the backup contains LBAC-protected data.

When backing up to tape, use of a variable block size is currently not supported. If you must use this option, ensure that you have well tested procedures in place that enable you to recover successfully, using backup images that were created with a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Snapshot backups should be complemented with regular disk backups in case of failure in the filer/storage system.

As you regularly backup your database, you might accumulate very large database backup images, many database logs and load copy images, all of which might be taking up a large amount of disk space. Refer to "Managing recovery objects" for information on how to manage these recovery objects.

You can use the **OPTIONS** parameter to enable backup operations in TSM environments supporting proxy nodes. For more information, see the "Configuring a Tivoli Storage Manager client" topic.

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the command returns additional information. The backup operation will return one result set, comprising one row per database partition that participated in the backup.

Table 29. Result set for a backup operation

Column name	Data type	Description
BACKUP_TIME	VARCHAR(14)	Corresponds to the timestamp string used to name the backup image.
DBPARTITIONNUM	SMALLINT	The database partition number on which the agent executed the backup operation.
SQLCODE	INTEGER	Final SQLCODE resulting from the backup processing on the specified database partition.
SQLERRMC	VARCHAR(70)	Final SQLERRMC resulting from the backup processing on the specified database partition.

Table 29. Result set for a backup operation (continued)

Column name	Data type	Description
SQLERRML	SMALLINT	Final SQLERRML resulting from the backup processing on the specified database partition.

If a non-partitioned database is backed up, or if a partitioned database is backed up using the traditional single-partition syntax, the result set will comprise a single row. DBPARTITIONNUM will contain the identifier number of the database partition being backed up.

SQLCODE, SQLERRMC, and SQLERRML refer to the equivalently-named members of the SQLCA that is returned by the backup on the specified database partition.

DESCRIBE command using the ADMIN_CMD procedure

The DESCRIBE command displays metadata about the columns, indexes, and data partitions of tables or views. This command can also display metadata about the output of SELECT, CALL, or XQuery statements.

Use the DESCRIBE command to display information about any of the following items:

- Output of a SELECT, CALL, or XQuery statement
- Columns of a table or a view
- Indexes of a table or a view
- Data partitions of a table or view

Authorization

The authorization required depends on the type of information you want to display using the DESCRIBE command.

- If the SYSTOOLSTMPSPACE table space exists, one of the authorities shown in the following table is required.

Object to display information about	Privileges or authorities required
Output of a SELECT statement or XQuery statement	Any of the following privileges or authorities for each table or view referenced in the SELECT statement: • SELECT privilege • DATAACCESS authority • DBADM authority • SQLADM authority • EXPLAIN authority
Output of a CALL statement	Any of the following privileges or authorities: • DATAACCESS authority • EXECUTE privilege on the stored procedure

Object to display information about	Privileges or authorities required
Columns of a table or a view	Any of the following privileges or authorities for the SYSCAT.COLUMNS system catalog table: • SELECT privilege • ACCESSCTRL authority • DATAACCESS authority • DBADM authority • SECADM authority • SQLADM authority If you want to use the **SHOW DETAIL** parameter, you also require any of these privileges or authorities on the SYSCAT.DATAPARTITIONEXPRESSION system catalog table. Because PUBLIC has all the privileges over declared temporary tables, you can use the command to display information about any declared temporary table that exists within your connection.
Indexes of a table or a view	Any of the following privileges or authorities on the SYSCAT.INDEXES system catalog table: • SELECT privilege • ACCESSCTRL authority • DATAACCESS authority • DBADM authority • SECADM authority • SQLADM authority If you want to use the **SHOW DETAIL** parameter, you also require EXECUTE privilege on the GET_INDEX_COLNAMES() UDF. Because PUBLIC has all the privileges over declared temporary tables, you can use the command to display information about any declared temporary table that exists within your connection.

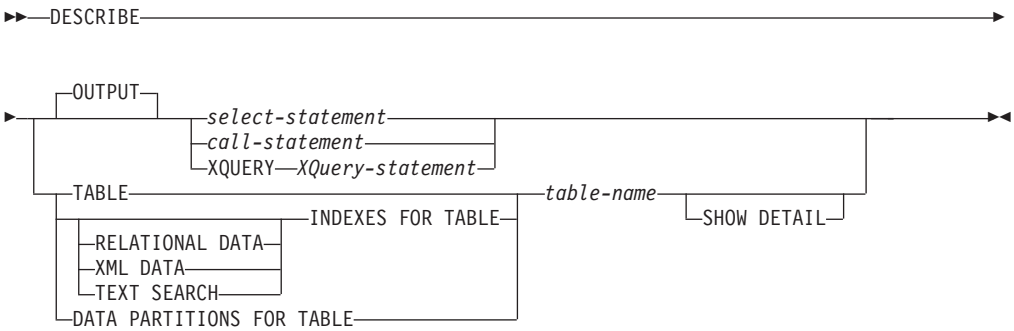
Object to display information about	Privileges or authorities required
Data partitions of a table or view	Any of the following privileges or authorities on the SYSCAT.DATAPARTITIONS system catalog table: • SELECT privilege • ACCESSCTRL authority • DATAACCESS authority • DBADM authority • SECADM authority • SQLADM authority Because PUBLIC has all the privileges over declared temporary tables, you can use the command to display information about any declared temporary table that exists within your connection.

- If the SYSTOOLSTMPSPACE table space does not exist, SYSADM or SYSCTRL authority is also required in addition to the one of the above authorities.

Required connection

Database

Command syntax



Command parameters

OUTPUT
Indicates that the output of the statement should be described. This keyword is optional.

select-statement | *call-statement* | **XQUERY** *XQuery-statement*
Identifies the statement about which information is wanted. The statement is automatically prepared by CLP. To identify an XQuery statement, precede the statement with the keyword **XQUERY**. A DESCRIBE OUTPUT statement only returns information about an implicitly hidden column if the column is explicitly specified as part of the SELECT list of the final result table of the query described.

TABLE *table-name*
Specifies the table or view to be described. The fully qualified name in the

form *schema.table-name* must be used. An alias for the table cannot be used in place of the actual table. Information about implicitly hidden columns is returned.

The DESCRIBE TABLE command lists the following information about each column:

- Column name
- Type schema
- Type name
- Length
- Scale
- Nulls (yes/no)

INDEXES FOR TABLE *table-name*

Specifies the table or view for which indexes need to be described. You can use the fully qualified name in the form *schema.table-name* or you can just specify the *table-name* and default schema will be used automatically. An alias for the table cannot be used in place of the actual table.

The DESCRIBE INDEXES FOR TABLE command lists the following information about each index of the table or view:

- Index schema
- Index name
- Unique rule
- Number of columns
- Index type

If the DESCRIBE INDEXES FOR TABLE command is specified with the **SHOW DETAIL** option, the index name is truncated when the index name is greater than 18 bytes. If no index type option is specified, information for all index types is listed: relational data index, index over XML data, and Text Search index. The output includes the following additional information:

- Index ID for a relational data index, an XML path index, an XML regions index, or an index over XML data
- Data Type for an index over XML data
- Hashed for an index over XML data
- Max VARCHAR Length for an index over XML data
- XML Pattern specified for an index over XML data
- Codepage for a text search index
- Language for a text search index
- Format specified for a text search index
- Update minimum for a text search index
- Update frequency for a text search index
- Collection directory for a text search index
- Column names

Specify an index type to list information for only a specific index type. Specifying multiple index types is not supported.

RELATIONAL DATA

If the RELATIONAL DATA index type option is specified without the **SHOW DETAIL** option, only the following information is listed:

- Index schema
- Index name
- Unique rule
- Number of columns

If **SHOW DETAIL** is specified, the column names information is also listed.

XML DATA

If the **XML DATA** index type option is specified without the **SHOW DETAIL** option, only the following information is listed:

- Index schema
- Index name
- Unique rule
- Number of columns
- Index type

If **SHOW DETAIL** is specified, the following information for an index over XML data is also listed:

- Index ID
- Data type
- Hashed
- Max Varchar length
- XML Pattern
- Column names

TEXT SEARCH

If the **TEXT SEARCH** index type option is specified without the **SHOW DETAIL** option, only the following information is listed:

- Index schema
- Index name

If **SHOW DETAIL** is specified, the following text search index information is also listed:

- Column name
- Codepage
- Language
- Format
- Update minimum
- Update frequency
- Collection directory

If the **TEXT SEARCH** option is specified and a text search option is not installed or not properly configured, an error (SQLSTATE 42724) is returned.

See DB2 Text Search for information listed in the columns.

DATA PARTITIONS FOR TABLE *table-name*

Specifies the table or view for which data partitions need to be described.

The information displayed for each data partition in the table includes; the partition identifier and the partitioning intervals. Results are ordered according to the partition identifier sequence. The fully qualified name in the form *schema.table-name* must be used. An alias for the table cannot be used in place of the actual table. The *schema* is the user name under which the table or view was created.

For the DESCRIBE DATA PARTITIONS FOR TABLE command, specifies that output include a second table with the following additional information:

- Data partition sequence identifier
- Data partition expression in SQL

SHOW DETAIL

For the DESCRIBE TABLE command, specifies that output include the following additional information as well as a second result set which contains the table data partition expressions (which might return 0 rows if the table is not data partitioned):

- Whether a CHARACTER, VARCHAR or LONG VARCHAR column was defined as FOR BIT DATA
- Column number
- Distribution key sequence
- Code page
- Default
- Table partitioning type (for tables partitioned by range this output appears below the original output)
- Partitioning key columns (for tables partitioned by range this output appears below the original output)
- Identifier of table space used for the index

Examples

Describing the output of a SELECT statement

The following example shows how to describe a SELECT statement:

```
CALL SYSPROC.ADMIN_CMD('describe select * from emp_photo')
```

The following is an example of output for this SELECT statement.

```
Result set 1
--------------

SQLTYPE_ID SQLTYPE      SQLLENGTH SQLSCALE SQLNAME_DATA    ...
---------- ---------..- --------- -------- ------------..- ...
       452 CHARACTER            6        0 EMPNO           ...
       448 VARCHAR             10        0 PHOTO_FORMAT    ...
       405 BLOB            102400        0 PICTURE         ...

  3 record(s) selected.

  Return Status = 0
```

Output for this SELECT statement (continued).

```
... SQLNAME_LENGTH SQLDATATYPENAME_DATA    SQLDATATYPENAME_LENGTH
... -------------- ---------------------..- ----------------------
...              5 SYSIBM  .CHARACTER                           18
...             12 SYSIBM  .VARCHAR                             16
...              7 SYSIBM  .BLOB                                13
```

Describing a table

Describing a non-partitioned table.

```
CALL SYSPROC.ADMIN_CMD('describe table org show detail')
```

The following is an example of output for this CALL statement.

```
Result set 1
--------------
COLNAME     TYPESCHEMA     TYPENAME     FOR_BINARY_DATA     ...
-------...- ----------...- ---------...- ---------------...- ...
DEPTNUMB    SYSIBM         SMALLINT     N                   ...
DEPTNAME    SYSIBM         VARCHAR      N                   ...
MANAGER     SYSIBM         SMALLINT     N                   ...
DIVISION    SYSIBM         VARCHAR      N                   ...
LOCATION    SYSIBM         VARCHAR      N                   ...

  5 record(s) selected.
```

Output for this CALL statement (continued).

```
... LENGTH SCALE NULLABLE COLNO PARTKEYSEQ CODEPAGE DEFAULT
... ------ ----- -------- ----- ---------- -------- -------
...      2     0 N            0          1        0 -
...     14     0 Y            1          0     1208 -
...      2     0 Y            2          0        0 -
...     10     0 Y            3          0     1208 -
...     13     0 Y            4          0     1208 -
```

Output for this CALL statement (continued).

```
Result set 2
--------------
DATA_PARTITION_KEY_SEQ DATA_PARTITION_EXPRESSION
---------------------- -------------------------

  0 record(s) selected.

Return Status = 0
```

Describing a partitioned table.

```
CALL SYSPROC.ADMIN_CMD('describe table part_table1 show detail')
```

The following is an example of output for this CALL statement.

```
Result set 1
--------------
COLNAME     TYPESCHEMA     TYPENAME FOR_BINARY_DATA ...
-------...- ----------...- -------- --------------- ...
COL1        SYSIBM         INTEGER  N               ...

  1 record(s) selected.
```

Output for this CALL statement (continued).

```
... LENGTH SCALE NULLABLE COLNO PARTKEYSEQ CODEPAGE DEFAULT
... ------ ----- -------- ----- ---------- -------- -------
...      4     0 N            0          1        0       -
```

Output for this CALL statement (continued).

```
Result set 2
--------------
DATA_PARTITION_KEY_SEQ DATA_PARTITION_EXPRESSION
---------------------- -------------------------
                     1 COL1

  1 record(s) selected
```

Describing a table index

The following example shows how to describe a table index. This call describes table USER1.DEPARTMENT and lists two relational data indexes, six xml data indexes, two text search indexes, and the system indexes:

```
CALL SYSPROC.ADMIN_CMD('describe indexes for table user1.department')
```

The following is an example of output for this CALL statement.

```
Result set 1
--------------

INDSCHEMA      INDNAME                UNIQUE_RULE
----------...  -------------------...  -------------------
SYSIBM         SQL070531145253450     DUPLICATES_ALLOWED
SYSIBM         SQL070531145253620     UNIQUE_ENTRIES_ONLY
USER1          RELIDX1                DUPLICATES_ALLOWED
USER1          RELIDX2                DUPLICATES_ALLOWED
SYSIBM         SQL070531145253650     PRIMARY_INDEX
USER1          XMLIDX1                DUPLICATES_ALLOWED
SYSIBM         SQL070531154625650     DUPLICATES_ALLOWED
USER1          XMLIDX2                DUPLICATES_ALLOWED
SYSIBM         SQL070531154626000     DUPLICATES_ALLOWED
USER1          XMLIDX3                DUPLICATES_ALLOWED
SYSIBM         SQL070531154626090     DUPLICATES_ALLOWED
USER1          XMLIDX4                DUPLICATES_ALLOWED
SYSIBM         SQL070531154626190     DUPLICATES_ALLOWED
USER1          XMLIDX5                DUPLICATES_ALLOWED
SYSIBM         SQL070531154626290     DUPLICATES_ALLOWED
USER1          XMLIDX6                DUPLICATES_ALLOWED
SYSIBM         SQL070531154626400     DUPLICATES_ALLOWED
USER1          TXTIDX1                -
USER1          TXTIDX2                -

  19 record(s) selected.

  Return Status = 0
```

Output for this CALL statement (continued).

```
...   COLCOUNT    INDEXTYPE
...   ---------   -------------------------
...           -   XML_DATA_REGIONS
...           1   XML_DATA_PATH
...           1   RELATIONAL_DATA
...           2   RELATIONAL_DATA
...           1   RELATIONAL_DATA
...           1   XML_DATA_VALUES_LOGICAL
...           1   XML_DATA_VALUES_PHYSICAL
...           1   XML_DATA_VALUES_LOGICAL
...           1   XML_DATA_VALUES_PHYSICAL
...           1   XML_DATA_VALUES_LOGICAL
...           1   XML_DATA_VALUES_PHYSICAL
...           1   XML_DATA_VALUES_LOGICAL
...           1   XML_DATA_VALUES_PHYSICAL
...           1   XML_DATA_VALUES_LOGICAL
...           1   XML_DATA_VALUES_PHYSICAL
```

```
...           1   XML_DATA_VALUES_LOGICAL
...           1   XML_DATA_VALUES_PHYSICAL
...           1   TEXT_SEARCH
...           1   TEXT_SEARCH
```

Describing a data partition

The following example shows how to describe data partitions.

```
CALL SYSPROC.ADMIN_CMD('describe data partitions for table part_table2')
```

The following is an example of output for this CALL statement.

```
Result set 1
--------------

DATA_PARTITION_ID LOW_KEY_INCLUSIVE LOW_KEY_VALUE ...
----------------- ----------------- ------------ ...
                0 Y                 1            ...
                1 Y                 10           ...
                2 Y                 20           ...

  3 record(s) selected.
```

Output for this CALL statement (continued).

```
... HIGH_KEY_INCLUSIVE HIGH_KEY_VALUE
... ------------------ --------------
... N                  10
... N                  20
... N                  40
```

The following example shows how to describe data partitions with 'SHOW DETAIL' clause.

```
CALL SYSPROC.ADMIN_CMD('describe data partitions
    for table part_table2 show detail')
```

The following is an example of output for this CALL statement.

```
Result set 1
--------------

DATA_PARTITION_ID LOW_KEY_INCLUSIVE LOW_KEY_VALUE ...
----------------- ----------------- ------------ ...
                0 Y                 1            ...
                1 Y                 10           ...
                2 Y                 20           ...

  3 record(s) selected.

Return Status = 0
```

Output for this CALL statement (continued).

```
... HIGH_KEY_INCLUSIVE HIGH_KEY_VALUE
... ------------------ --------------
... N                  10
... N                  20
... N                  40
```

Output for this CALL statement (continued).

```
Result set 2
--------------

DATA_PARTITION_ID DATA_PARTITION_NAME TBSPID ...
----------------- ------------------- ------ ...
```

```
                0 PART0                   3 ...
                1 PART1                   3 ...
                2 PART2                   3 ...

  3 record(s) selected.

Return Status = 0
```

Output for this CALL statement (continued).

```
... PARTITION_OBJECT_ID LONG_TBSPID ACCESSMODE  STATUS
... ------------------- ----------- ----------- ------
...                  15           3 FULL_ACCESS
...                  16           3 FULL_ACCESS
...                  17           3 FULL_ACCESS
```

Usage note

If the DESCRIBE command tries to create a temporary table and fails, creation of SYSTOOLSTMPSPACE is attempted, and then creation of the temporary table is attempted again, this time in SYSTOOLSTMPSPACE. SYSCTRL or SYSADM authority is required to create the SYSTOOLSTMPSPACE table space.

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the commands return additional information in result sets as follows:

- Table 30: DESCRIBE *select-statement*, DESCRIBE *call-statement* and DESCRIBE XQUERY *XQuery-statement* commands
- Table 31 on page 61: Result set 1 for the DESCRIBE TABLE command
- Table 32 on page 62: Result set 2 for the DESCRIBE TABLE command
- Table 33 on page 62: DESCRIBE INDEXES FOR TABLE command
- Table 34 on page 63: Result set 1 for the DESCRIBE DATA PARTITIONS FOR TABLE command
- Table 35 on page 64: Result set 2 for the DESCRIBE DATA PARTITIONS FOR TABLE command

Table 30. Result set returned by the DESCRIBE select-statement, DESCRIBE call-statement and DESCRIBE XQUERY XQuery-statement commands

Column name	Data type	LOB only[1]	Description
SQLTYPE_ID	SMALLINT	No	Data type of the column, as it appears in the SQLTYPE field of the SQL descriptor area (SQLDA).
SQLTYPE	VARCHAR (257)	No	Data type corresponding to the SQLTYPE_ID value.
SQLLEN	INTEGER	No	Length attribute of the column, as it appears in the SQLLEN field of the SQLDA.
SQLSCALE	SMALLINT	No	Number of digits in the fractional part of a decimal value; 0 in the case of other data types.
SQLNAME_DATA	VARCHAR (128)	No	Name of the column.
SQLNAME_LENGTH	SMALLINT	No	Length of the column name.
SQLDATA_TYPESCHEMA	VARCHAR (128)	Yes	Data type schema name.

Table 30. Result set returned by the DESCRIBE select-statement, DESCRIBE call-statement and DESCRIBE XQUERY XQuery-statement commands (continued)

Column name	Data type	LOB only[1]	Description
SQLDATA_TYPENAME	VARCHAR (128)	Yes	Data type name.

Note: [1]: Yes indicates that non-null values are returned only when there is LOB data being described.

Table 31. Result set 1 returned by the DESCRIBE TABLE command

Column name	Data type	Detail[2]	Description
COLNAME	VARCHAR (128)	No	Column name.
TYPESCHEMA	VARCHAR (128)	No	If the column name is distinct, the schema name is returned, otherwise, 'SYSIBM' is returned.
TYPENAME	VARCHAR (128)	No	Name of the column type.
FOR_BINARY_DATA	CHAR (1)	Yes	Returns 'Y' if the column is of type CHAR, VARCHAR or LONG VARCHAR, and is defined as FOR BIT DATA, 'N' otherwise.
LENGTH	INTEGER	No	Maximum length of the data. For DECIMAL data, this indicates the precision. For discinct types, 0 is returned.
SCALE	SMALLINT	No	For DECIMAL data, this indicates the scale. For all other types, 0 is returned.
NULLABLE	CHAR (1)	No	One of: • 'Y' if column is nullable • 'N' if column is not nullable
COLNO	SMALLINT	Yes	Ordinal of the column.
PARTKEYSEQ	SMALLINT	Yes	Ordinal of the column within the table's partitioning key. NULL or 0 is returned if the column is not part of the partitioning key, and is NULL for subtables and hierarchy tables.
CODEPAGE	SMALLINT	Yes	Code page of the column and is one of: • Value of the database code page for columns that are not defined with FOR BIT DATA. • Value of the DBCS code page for graphic columns. • 0 otherwise.
DEFAULT	VARCHAR (254)	Yes	Default value for the column of a table expressed as a constant, special register, or cast-function appropriate for the data type of the column. Might also be NULL.

Note: [2]: Yes indicates that non-null values are returned only when the SHOW DETAIL clause is used.

Table 32. Result set 2 returned by the DESCRIBE TABLE command when the SHOW DETAIL clause is used.

Column name	Data type	Description
DATA_PARTITION_KEY_SEQ	INTEGER	Data partition key number, for example, 1 for the first data partition expression and 2 for the second data partition expression.
DATA_PARTITION_EXPRESSION	CLOB (32K)	Expression for this data partition key in SQL syntax

Table 33. Result set returned by the DESCRIBE INDEXES FOR TABLE command

Column name	Data type	Detail[3]	Index type option[4, 5]	Description
INDSCHEMA	VARCHAR (128)	No	RELATIONAL DATA XML DATA TEXT SEARCH	Index schema name.
INDNAME	VARCHAR (128)	No	RELATIONAL DATA XML DATA TEXT SEARCH	Index name.
UNIQUE_RULE	VARCHAR (30)	No	RELATIONAL DATA XML DATA	One of following values: • DUPLICATES_ALLOWED • PRIMARY_INDEX • UNIQUE_ENTRIES_ONLY
INDEX _PARTITIONING	CHAR(1)	No	N/A	Identifies the partitioning characteristic of the index. Possible values are: • N= Nonpartitioned index • P= Partitioned index • Blank = Index is not on a partitioned table
COLCOUNT	SMALLINT	No	RELATIONAL DATA XML DATA	Number of columns in the key, plus the number of include columns, if any.
INDEX_TYPE	VARCHAR (30)	No	RELATIONAL DATA XML DATA TEXT SEARCH	Type of index: • RELATIONAL_DATA • TEXT_SEARCH • XML_DATA_REGIONS • XML_DATA_PATH • XML_DATA_VALUES_LOGICAL • XML_DATA_VALUES_PHYSICAL
INDEX_ID	SMALLINT	Yes	RELATIONAL DATA XML DATA	Index ID for a relational data index, an XML path index, an XML regions index, or an index over XML data
DATA_TYPE	VARCHAR (128)	Yes	XML DATA	SQL data type specified for an index over XML data. One of the following values: • VARCHAR • DOUBLE • DATE • TIMESTAMP

Table 33. Result set returned by the DESCRIBE INDEXES FOR TABLE command (continued)

Column name	Data type	Detail[3]	Index type option[4, 5]	Description
HASHED	CHAR (1)	Yes	XML DATA	Indicates whether or not the value for an index over XML data is hashed. • 'Y' if the value is hashed. • 'N' if the value is not hashed.
LENGTH	SMALLINT	Yes	XML DATA	For an index over XML data, the VARCHAR (*integer*) length; 0 otherwise.
PATTERN	CLOB (2M)	Yes	XML DATA	XML pattern expression specified for an index over XML data
CODEPAGE	INTEGER	Yes	TEXT SEARCH	Document code page specified for the text search index
LANGUAGE	VARCHAR (5)	Yes	TEXT SEARCH	Document language specified for the text search index
FORMAT	VARCHAR (30)	Yes	TEXT SEARCH	Document format specified for a text search index
UPDATEMINIMUM	INTEGER	Yes	TEXT SEARCH	Minimum number of entries in the text search log table before an incremental update is performed
UPDATEFREQUENCY	VARCHAR (300)	Yes	TEXT SEARCH	Trigger criterion specified for applying updates to the text index
COLLECTION DIRECTORY	VARCHAR (512)	Yes	TEXT SEARCH	Directory specified for the text search index files
COLNAMES	VARCHAR (2048)	Yes	RELATIONAL DATA XML DATA TEXT SEARCH	List of the column names, each preceded with a + to indicate ascending order or a - to indicate descending order.

Note: [3]: Yes indicates that values are returned only when the SHOW DETAIL clause is used without specifying an index type option. Values might be NULL.

Note: [4]: Indicates the values returned when using DESCRIBE *index-type* INDEXES FOR TABLE. For example, INDEX_ID values are not returned if TEXT SEARCH is specified as *index-type*. INDEX_ID values are returned if either RELATIONAL DATA or XML DATA are specified.

Note: [5]: When using DESCRIBE *index-type* INDEXES FOR TABLE SHOW DETAIL, the values are returned only when the index type is listed. For example, DATA_TYPE values are returned if XML DATA is specified as *index-type*. DATA_TYPE values are not returned if either TEXT SEARCH or RELATIONAL DATA is specified as *index-type*.

Table 34. Result set 1 returned by the DESCRIBE DATA PARTITIONS FOR TABLE command

Column name	Data type	Detail[2]	Description
DATA_PARTITION_ID	INTEGER	No	Data partition identifier.
LOW_KEY_INCLUSIVE	CHAR (1)	No	'Y' if the low key value is inclusive, otherwise, 'N'.
LOW_KEY_VALUE	VARCHAR (512)	No	Low key value for this data partition.

Table 34. Result set 1 returned by the DESCRIBE DATA PARTITIONS FOR TABLE command (continued)

Column name	Data type	Detail[2]	Description
HIGH_KEY_INCLUSIVE	CHAR (1)	No	'Y' if the high key value is inclusive, otherwise, 'N'.
HIGH_KEY_VALUE	VARCHAR (512)	No	High key value for this data partition.

Note: [2]: Yes indicates that non-null values are returned only when the SHOW DETAIL clause is used.

Table 35. Result set 2 returned by the DESCRIBE DATA PARTITIONS FOR TABLE command when the SHOW DETAIL clause is used.

Column name	Data type	Description
DATA_PARTITION_ID	INTEGER	Data partition identifier.
DATA_PARTITION_NAME	VARCHAR (128)	Data partition name.
TBSPID	INTEGER	Identifier of the table space where this data partition is stored.
PARTITION_OBJECT_ID	INTEGER	Identifier of the DMS object where this data partition is stored.
LONG_TBSPID	INTEGER	Identifier of the table space where long data is stored.
INDEX_TBSPID	INTEGER	Identifier of the table space where index data is stored.
ACCESSMODE	VARCHAR (20)	Defines accessibility of the data partition and is one of: • FULL_ACCESS • NO_ACCESS • NO_DATA_MOVEMENT • READ_ONLY
STATUS	VARCHAR(64)	Data partition status and can be one of: • NEWLY_ATTACHED • NEWLY_DETACHED: MQT maintenance is required. • INDEX_CLEANUP_PENDING: detached data partition whose tuple in SYSDATAPARTITIONS is maintained only for index cleanup. This tuple is removed when all index records referring to the detached data partition have been deleted. The column is blank otherwise.

DROP CONTACT command using the ADMIN_CMD procedure

Removes a contact from the list of contacts defined on the local system. A contact is a user to whom the Scheduler and Health Monitor send messages. The setting of the Database Administration Server (DAS) **contact_host** configuration parameter determines whether the list is local or global.

Authorization

None

Required connection

Database. The DAS must be running.

Command syntax

Command parameters

CONTACT *name*
The name of the contact that will be dropped from the local system.

Example

Drop the contact named 'testuser' from the list of contacts on the server system.

```
CALL SYSPROC.ADMIN_CMD( 'drop contact testuser' )
```

Usage notes

The DAS must have been created and be running.

Command execution status is returned in the SQLCA resulting from the CALL statement.

DROP CONTACTGROUP command using the ADMIN_CMD procedure

Removes a contact group from the list of contacts defined on the local system. A contact group contains a list of users to whom the Scheduler and Health Monitor send messages. The setting of the Database Administration Server (DAS) **contact_host** configuration parameter determines whether the list is local or global.

Authorization

None

Required Connection

Database. The DAS must be running.

Command Syntax

Command Parameters

CONTACTGROUP *name*
The name of the contact group that will be dropped from the local system.

Example

Drop the contact group named 'gname1'.

```
CALL SYSPROC.ADMIN_CMD( 'drop contactgroup gname1' )
```

Usage notes

The DAS must have been created and be running.

Command execution status is returned in the SQLCA resulting from the CALL statement.

EXPORT command using the ADMIN_CMD procedure

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables. The data is exported to the server only.

Quick link to "File type modifiers for the export utility" on page 72.

Authorization

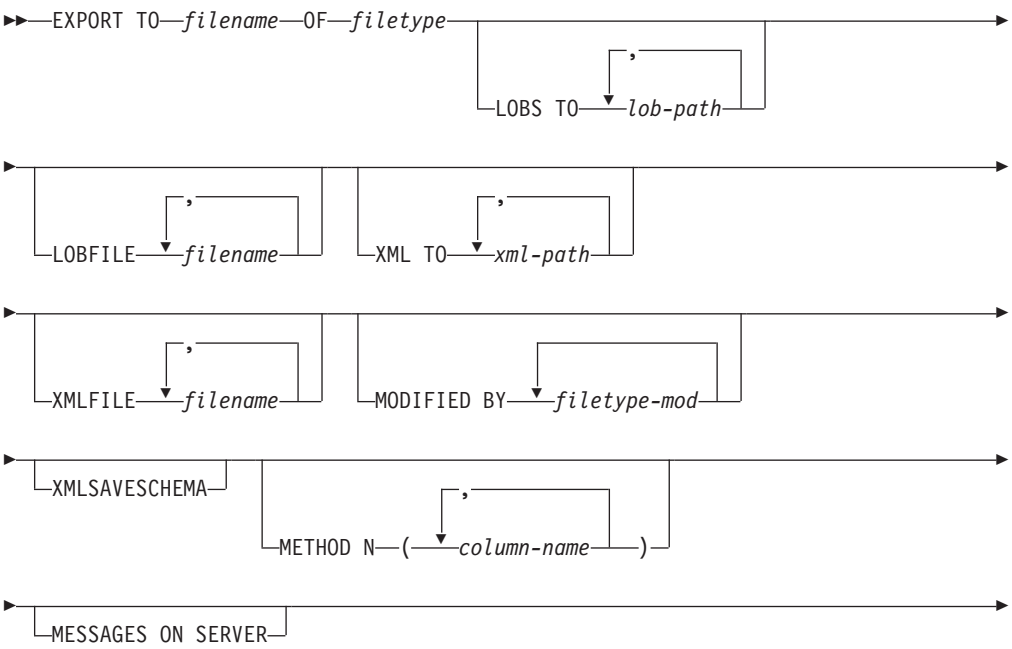
One of the following:

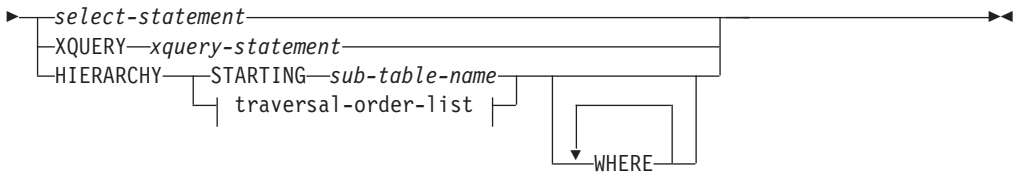
- DATAACCESS authority
- CONTROL or SELECT privilege on each participating table or view

Required connection

Database. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect™ gateway or loop back environment.

Command syntax





traversal-order-list:

Command parameters

HIERARCHY *traversal-order-list*
Export a sub-hierarchy using the specified traverse order. All sub-tables must be listed in PRE-ORDER fashion. The first sub-table name is used as the target table name for the SELECT statement.

HIERARCHY STARTING *sub-table-name*
Using the default traverse order (OUTER order for ASC, DEL, or WSF files, or the order stored in PC/IXF data files), export a sub-hierarchy starting from *sub-table-name*.

LOBFILE *filename*
Specifies one or more base file names for the LOB files. When name space is exhausted for the first name, the second name is used, and so on. This will implicitly activate the `LOBSINFILE` behavior.

When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *lob-path*), and then appending a 3-digit sequence number to start and the three character identifier `lob`. For example, if the current LOB path is the directory `/u/foo/lob/path/`, and the current LOB file name is `bar`, the LOB files created will be `/u/foo/lob/path/bar.001.lob`, `/u/foo/lob/path/bar.002.lob`, and so on. The 3-digit sequence number in the LOB file name will grow to 4-digits once 999 is used, 4-digits will grow to 5-digits once 9999 is used, and so on.

LOBS TO *lob-path*
Specifies one or more paths to directories in which the LOB files are to be stored. The path(s) must exist on the coordinator partition of the server and must be fully qualified. There will be at least one file per LOB path, and each file will contain at least one LOB. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

MESSAGES ON SERVER
Specifies that the message file created on the server by the EXPORT command is to be saved. The result set returned will include the following two columns: MSG_RETRIEVAL, which is the SQL statement required to retrieve all the warnings and error messages that occur during this operation, and MSG_REMOVAL, which is the SQL statement required to clean up the messages.

If this clause is not specified, the message file will be deleted when the ADMIN_CMD procedure returns to the caller. The MSG_RETRIEVAL and MSG_REMOVAL column in the result set will contain null values.

Note that with or without the clause, the fenced user ID must have the authority to create files under the directory indicated by the **DB2_UTIL_MSGPATH** registry variable, as well as the directory where the data is to be exported to.

METHOD N *column-name*
Specifies one or more column names to be used in the output file. If this parameter is not specified, the column names in the table are used. This parameter is valid only for WSF and IXF files, but is not valid when exporting hierarchical data.

MODIFIED BY *filetype-mod*
Specifies file type modifier options. See "File type modifiers for the export utility" on page 72.

OF *filetype*
Specifies the format of the data in the output file:

- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs.
- WSF (work sheet format), which is used by programs such as:
  - Lotus® 1-2-3®
  - Lotus Symphony™

  When exporting BIGINT or DECIMAL data, only values that fall within the range of type DOUBLE can be exported accurately. Although values that do not fall within this range are also exported, importing or loading these values back might result in incorrect data, depending on the operating system.

  Note: Support for the WSF file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.
- IXF (Integration Exchange Format, PC version) is a proprietary binary format.

select-statement
Specifies the SELECT or XQUERY statement that will return the data to be exported. If the statement causes an error, a message is written to the message file (or to standard output). If the error code is one of SQL0012W, SQL0347W, SQL0360W, SQL0437W, or SQL1824W, the export operation continues; otherwise, it stops.

TO *filename*
Specifies the name of the file to which data is to be exported to on the server. This must be a fully qualified path and must exist on the server coordinator partition.

If the name of a file that already exists is specified, the export utility overwrites the contents of the file; it does not append the information.

XMLFILE *filename*
Specifies one or more base file names for the XML files. When name space is exhausted for the first name, the second name is used, and so on.

When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the

current path (from *xml-path*), appending a 3-digit sequence number, and appending the three character identifier `xml`. For example, if the current XML path is the directory `/u/foo/xml/path/`, and the current XML file name is `bar`, the XML files created will be `/u/foo/xml/path/bar.001.xml`, `/u/foo/xml/path/bar.002.xml`, and so on.

XML TO *xml-path*
Specifies one or more paths to directories in which the XML files are to be stored. There will be at least one file per XML path, and each file will contain at least one XQuery Data Model (XDM) instance. If more than one path is specified, then XDM instances are distributed evenly among the paths.

XMLSAVESCHEMA
Specifies that XML schema information should be saved for all XML columns. For each exported XML document that was validated against an XML schema when it was inserted, the fully qualified SQL identifier of that schema will be stored as an (SCH) attribute inside the corresponding XML Data Specifier (XDS). If the exported document was not validated against an XML schema or the schema object no longer exists in the database, an SCH attribute will not be included in the corresponding XDS.

The schema and name portions of the SQL identifier are stored as the "OBJECTSCHEMA" and "OBJECTNAME" values in the row of the SYSCAT.XSROBJECTS catalog table corresponding to the XML schema.

The **XMLSAVESCHEMA** option is not compatible with XQuery sequences that do not produce well-formed XML documents.

Example

The following example shows how to export information from the STAFF table in the SAMPLE database to the file `myfile.ixf`. The output will be in IXF format. You must be connected to the SAMPLE database before issuing the command.

```
CALL SYSPROC.ADMIN_CMD ('EXPORT to /home/user1/data/myfile.ixf
   OF ixf MESSAGES ON SERVER select * from staff')
```

Usage notes

- Any path used in the EXPORT command must be a valid fully-qualified path on the server.
- If a table contains LOB columns, at least one fully-qualified LOB path and LOB name must be specified, using the **LOBS TO** and **LOBFILE** clauses.
- The export utility issues a COMMIT statement at the beginning of the operation which, in the case of Type 2 connections, causes the procedure to return SQL30090N with reason code 2.
- When exporting from a UCS-2 database to a delimited ASCII (DEL) file, all character data is converted to the code page that is in effect where the procedure is executing. Both character string and graphic string data are converted to the same SBCS or MBCS code page of the server.
- Be sure to complete all table operations and release all locks before starting an export operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.
- Table aliases can be used in the SELECT statement.
- The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.

- The export utility produces a warning message whenever a character column with a length greater than 254 is selected for export to DEL format files.
- PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.
- The file copying step is not necessary if the source and the target databases are both accessible from the same client.
- DB2 Connect can be used to export tables from DRDA® servers such as DB2 for OS/390®, DB2 for VM and VSE, and DB2 for OS/400®. Only PC/IXF export is supported.
- When exporting to the IXF format, if identifiers exceed the maximum size supported by the IXF format, the export will succeed but the resulting datafile cannot be used by a subsequent import operation using the CREATE mode. SQL27984W will be returned.
- When exporting to a diskette on Windows, and the table that has more data than the capacity of a single diskette, the system will prompt for another diskette, and multiple-part PC/IXF files (also known as multi-volume PC/IXF files, or logically split PC/IXF files), are generated and stored in separate diskettes. In each file, with the exception of the last, there is a DB2 CONTINUATION RECORD (or "AC" Record in short) written to indicate the files are logically split and where to look for the next file. The files can then be transferred to an AIX system, to be read by the import and load utilities. The export utility will not create multiple-part PC/IXF files when invoked from an AIX system. For detailed usage, see the IMPORT command or LOAD command.
- The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form `SELECT * FROM tablename`.
- When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the **WHERE** clause. Fullselect and *select-statement* cannot be specified when exporting a hierarchy.
- For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.
- Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.
- When exporting data from a table that has protected rows, the LBAC credentials held by the session authorization id might limit the rows that are exported. Rows that the session authorization ID does not have read access to will not be exported. No error or warning is given.
- If the LBAC credentials held by the session authorization id do not allow reading from one or more protected columns included in the export then the export fails and an error (SQLSTATE 42512) is returned.
- When running Data Movement utilities such as export and db2move, the query compiler might determine that the underlying query will run more efficiently against an MQT than the base table or tables. In this case, the query will execute against a refresh deferred MQT, and the result of the utilities might not accurately represent the data in the underlying table.
- Export packages are bound using `DATETIME ISO` format, thus, all date/time/timestamp values are converted into ISO format when cast to a string representation. Since the CLP packages are bound using `DATETIME LOC` format

(locale specific format), you may see inconsistent behavior between CLP and export if the CLP DATETIME format is different from ISO. For instance, the following SELECT statement may return expected results:

```
db2 select col2 from tab1 where char(col2)='05/10/2005';
   COL2
   ----------
   05/10/2005
   05/10/2005
   05/10/2005
   3 record(s) selected.
```

But an export command using the same select clause will not:

```
db2 export to test.del of del select col2 from test
where char(col2)='05/10/2005';
   Number of rows exported: 0
```

Now, replacing the LOCALE date format with ISO format gives the expected results:

```
db2 export to test.del of del select col2 from test
where char(col2)='2005-05-10';
   Number of rows exported: 3
```

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the command returns additional information in result sets as follows:

Table 36. Result set returned by the EXPORT command

Column name	Data type	Description
ROWS_EXPORTED	BIGINT	Total number of exported rows.

MSG_RETRIEVAL	VARCHAR(512)	SQL statement that is used to retrieve messages created by this utility. For example:
		`SELECT SQLCODE, MSG`
		`FROM TABLE (SYSPROC.ADMIN_GET_MSGS`
		`('3203498_txu')) AS MSG`

MSG_REMOVAL	VARCHAR(512)	SQL statement that is used to clean up messages created by this utility. For example:
		`CALL SYSPROC.ADMIN_REMOVE_MSGS`
		`('3203498_txu')`

File type modifiers for the export utility

Table 37. Valid file type modifiers for the export utility: All file formats

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file `db2exp.001`, and is 456 bytes long. If you specify the **lobsinfile** modifier when using EXPORT, the LOB data is placed in the locations specified by the **LOBS TO** clause. Otherwise the LOB data is sent to the data file directory. The **LOBS TO** clause specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The **LOBS TO** or **LOBFILE** options will implicitly activate the LOBSINFILE behavior. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
xmlinsepfiles	Each XQuery Data Model (XDM) instance is written to a separate file. By default, multiple values are concatenated together in the same file.
lobsinsepfiles	Each LOB value is written to a separate file. By default, multiple values are concatenated together in the same file.
xmlnodeclaration	XDM instances are written without an XML declaration tag. By default, XDM instances are exported with an XML declaration tag at the beginning that includes an encoding attribute.
xmlchar	XDM instances are written in the character codepage. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, XDM instances are written out in Unicode.
xmlgraphic	If the `xmlgraphic` modifier is specified with the EXPORT command, the exported XML document will be encoded in the UTF-16 code page regardless of the application code page or the `codepage` file type modifier.

Table 38. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[2] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	
	The single quotation mark (') can also be specified as a character string delimiter as follows:
	`modified by chardel''`

Table 38. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data to this code page from the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[2]
	
	In the following example, `coldel;` causes the export utility to use the semicolon character (;) as a column delimiter for the exported data:
	`db2 "export to temp of del modified by coldel;`
	`select * from staff where dept = 20"`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[2]
nochardel	Column data will not be surrounded by character delimiters. This option should not be specified if the data is intended to be imported or loaded using DB2. It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.[2]

striplzeros	Removes the leading zeros from all exported decimal columns.
	
	Consider the following example:
	`db2 create table decimalTable ( c1 decimal( 31, 2 ) )`
	`db2 insert into decimalTable values ( 1.1 )`
	
	`db2 export to data of del select * from decimalTable`
	
	`db2 export to data of del modified by STRIPLZEROS`
	`select * from decimalTable`
	
	In the first export operation, the content of the exported file data will be +0000000000000000000000000000001.10. In the second operation, which is identical to the first except for the `striplzeros` modifier, the content of the exported file data will be +1.10.

Table 38. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[4] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`U (1 to 12 times) - Fractional seconds(number of occurrences of U represent the number of digits with each digit ranging from 0 to 9`
	`TT - Meridian indicator (AM or PM)`
	Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The MMM element will produce the following values: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', and 'Dec'. 'Jan' is equal to month 1, and 'Dec' is equal to month 12.
	The following example illustrates how to export data containing user-defined time stamp formats from a table called 'schedule':
	`db2 export to delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`select * from schedule`

Table 39. Valid file type modifiers for the export utility: IXF file format

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data from this code page to the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive.

Table 40. Valid file type modifiers for the export utility: WSF file format[6]

Modifier	Description
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[5] This is the default.

Table 40. Valid file type modifiers for the export utility: WSF file format[6] (continued)

Modifier	Description
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[5]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[5]
4	Creates a WSF file containing DBCS characters.

Note:

1. The export utility does not issue a warning if an attempt is made to use unsupported file types with the **MODIFIED BY** option. If this is attempted, the export operation fails, and an error code is returned.
2. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
3. The export utility normally writes
   - date data in *YYYYMMDD* format
   - char(date) data in "*YYYY-MM-DD*" format
   - time data in "*HH.MM.SS*" format
   - time stamp data in "*YYYY-MM-DD-HH.MM.SS.uuuuuu*" format

   Data contained in any datetime columns specified in the SELECT statement for the export operation will also be in these formats.
4. For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```
5. These files can also be directed to a specific product by specifying an `L` for Lotus 1-2-3, or an `S` for Symphony in the *filetype-mod* parameter string. Only one value or product designator can be specified. Support for the WSF file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.
6. The WSF file format is not supported for XML columns. Support for this file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.
7. All XDM instances are written to XML files that are separate from the main data file, even if neither the **XMLFILE** nor the **XML TO** clause is specified. By default, XML files are written to the path of the exported data file. The default base name for XML files is the name of the exported data file with the extension ".xml" appended to it.

8. All XDM instances are written with an XML declaration at the beginning that includes an encoding attribute, unless the XMLNODECLARATION file type modifier is specified.
9. By default, all XDM instances are written in Unicode unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. The default path for XML data and LOB data is the path of the main data file. The default XML file base name is the main data file. The default LOB file base name is the main data file. For example, if the main data file is:

    ```
    /mypath/myfile.del
    ```

    the default path for XML data and LOB data is:

    ```
    /mypath"
    ```

    the default XML file base name is:

    ```
    myfile.del
    ```

    and the default LOB file base name is:

    ```
    myfile.del
    ```

    The LOBSINFILE file type modifier must be specified in order to have LOB files generated.
11. The export utility appends a numeric identifier to each LOB file or XML file. The identifier starts as a 3 digit, 0 padded sequence value, starting at:

    ```
    .001
    ```

    After the 999th LOB file or XML file, the identifier will no longer be padded with zeroes (for example, the 1000th LOG file or XML file will have an extension of:

    ```
    .1000
    ```

    Following the numeric identifier is a three character type identifier representing the data type, either:

    ```
    .lob
    ```

    or

    ```
    .xml
    ```

    For example, a generated LOB file would have a name in the format:

    ```
    myfile.del.001.lob
    ```

    and a generated XML file would be have a name in the format:

    ```
    myfile.del.001.xml
    ```
12. It is possible to have the export utility export XDM instances that are not well-formed documents by specifying an XQuery. However, you will not be able to import or load these exported documents directly into an XML column, since XML columns can only contain complete documents.

FORCE APPLICATION command using the ADMIN_CMD procedure

Forces local or remote users or applications off the system to allow for maintenance on a server.

Attention: If an operation that cannot be interrupted (RESTORE DATABASE, for example) is forced, the operation must be successfully re-executed before the database becomes available.

Scope

This command affects all database partitions that are listed in the `$HOME/sqllib/db2nodes.cfg` file.

In a partitioned database environment, this command does not have to be issued from the coordinator database partition of the application being forced. It can be issued from any node (database partition server) in the partitioned database environment.

Authorization

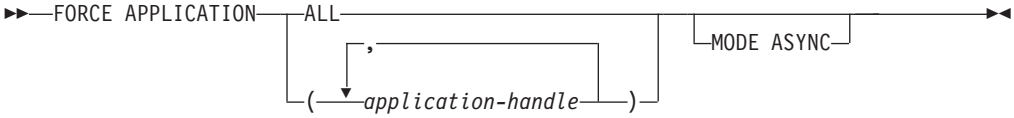
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT

Required connection

Database

Command syntax



Command parameters

FORCE APPLICATION

ALL All applications will be disconnected from the database. This might close the connection the ADMIN_CMD procedure is running on, which causes an SQL1224N error to be returned for the ADMIN_CMD procedure once the force operation is completed successfully.

application-handle
Specifies the agent to be terminated. List the values using the LIST APPLICATIONS command.

MODE ASYNC
The command does not wait for all specified users to be terminated before returning; it returns as soon as the function has been successfully issued or an error (such as invalid syntax) is discovered.

This is the only mode that is currently supported.

Examples

The following example forces two users, with *application-handle* values of 41408 and 55458, to disconnect from the database:

```
CALL SYSPROC.ADMIN_CMD( 'force application ( 41408, 55458 )' )
```

Usage notes

The database manager remains active so that subsequent database manager operations can be handled without the need for db2start.

To preserve database integrity, only users who are idling or executing interruptible database operations can be terminated.

The following types of users and applications cannot be forced:

- users creating a database
- system applications

In order to successfully force these types of users and applications, the database must be deactivated and/or the instance restarted.

After a FORCE APPLICATION has been issued, the database will still accept requests to connect. Additional forces might be required to completely force all users off.

Command execution status is returned in the SQLCA resulting from the CALL statement.

GET STMM TUNING DBPARTITIONNUM command using the ADMIN_CMD procedure

Used to read the catalog tables to report the user preferred self tuning memory manager (STMM) tuning database partition number and current STMM tuning database partition number.

Authorization

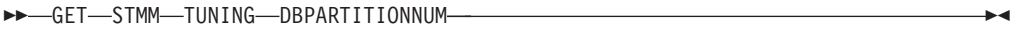
The privileges held by the authorization ID of the statement must include at least one of the following authorities or privilege:

- DBADM
- SECADM
- SQLADM
- ACCESSCTRL
- DATAACCESS
- SELECT on SYSIBM.SYSTUNINGINFO

Required connection

Database

Command syntax



Example

```
CALL SYSPROC.ADMIN_CMD( 'get stmm tuning dbpartitionnum' )
```

The following is an example of output from this query.

```
Result set 1
  --------------

  USER_PREFERRED_NUMBER CURRENT_NUMBER
  --------------------- --------------
                      2              2

  1 record(s) selected.

  Return Status = 0
```

Usage notes

The user preferred self tuning memory manager (STMM) tuning database partition number (USER_PREFERRED_NUMBER) is set by the user and specifies the database partition on which the user wishes to run the memory tuner. While the database is running, the tuning partition is updated asynchronously a few times an hour. As a result, it is possible that the CURRENT_NUMBER and USER_PREFERRED_NUMBER returned are not in sync after an update of the user preferred STMM partition number. To resolve this, either wait for the CURRENT_NUMBER to be updated asynchronously, or stop and start the database to force the update of CURRENT_NUMBER.

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the command returns additional information in the following result set:

Table 41. Result set returned by the GET STMM TUNING DBPARTITIONNUM command

Column name	Data type	Description
USER_PREFERRED_NUMBER	INTEGER	User preferred self tuning memory manager (STMM) tuning database partition number. A value of -1 indicates that the default database partition is used.
CURRENT_NUMBER	INTEGER	Current STMM tuning database partition number. A value of -1 indicates that the default database partition is used.

IMPORT command using the ADMIN_CMD procedure

Inserts data from an external file with a supported file format into a table, hierarchy, view or nickname. LOAD is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Quick link to “File type modifiers for the import utility” on page 94.

Authorization

- IMPORT using the **INSERT** option requires one of the following:
  - DATAACCESS authority

  - CONTROL privilege on each participating table, view, or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the **INSERT_UPDATE** option, requires one of the following:
  - DATAACCESS authority
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the **REPLACE** or **REPLACE_CREATE** option, requires one of the following:
  - DATAACCESS authority
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the **CREATE** or **REPLACE_CREATE** option, requires one of the following:
  - DBADM authority
  - CREATETAB authority on the database and USE privilege on the table space, as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- IMPORT to a hierarchy that does not exist using the **CREATE**, or the **REPLACE_CREATE** option, requires one of the following:
  - DBADM authority
  - CREATETAB authority on the database and USE privilege on the table space and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the **REPLACE_CREATE** option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the **REPLACE** option requires one of the following:
  - DATAACCESS authority
  - CONTROL privilege on every sub-table in the hierarchy
- To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the import fails and an error (SQLSTATE 42512) is returned.
- To import data into a table that has protected rows, the session authorization ID must hold LBAC credentials that meet these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access

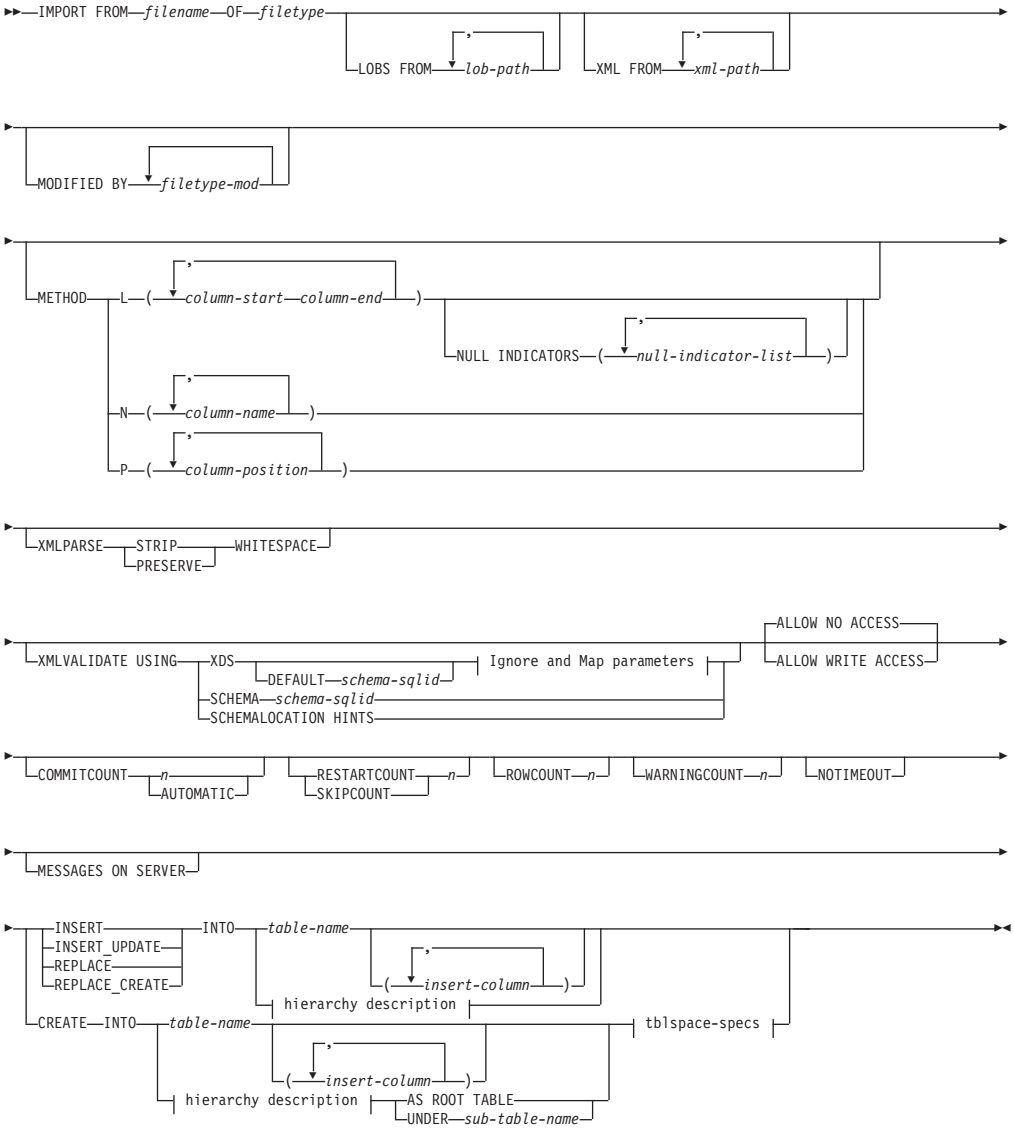
  The label on the row to insert, the user's LBAC credentials, the security policy definition, and the LBAC rules determine the label on the row.
- If the **REPLACE** or **REPLACE_CREATE** option is specified, the session authorization ID must have the authority to drop the table.

- To import data into a nickname, the session authorization ID must have the privilege to access and use a specified data source in pass-through mode.

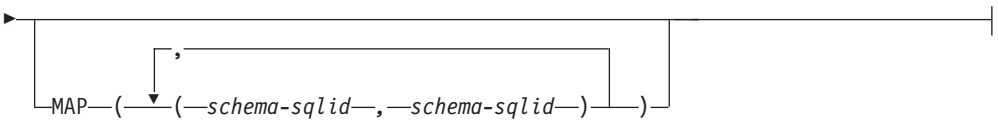
Required connection

Database. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect gateway or loop back environment.

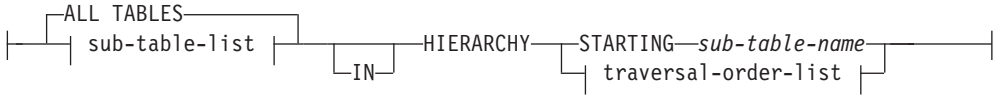
Command syntax



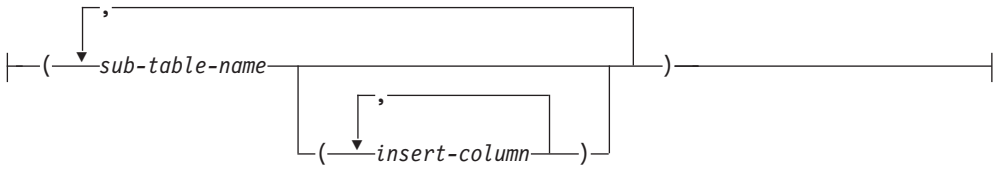
Ignore and Map parameters:



hierarchy description:

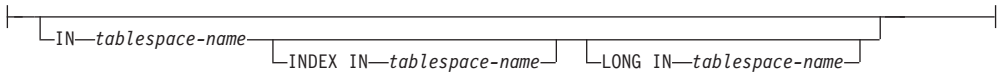


sub-table-list:



traversal-order-list:

tblspace-specs:



Command parameters

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal order.

ALLOW NO ACCESS
Runs import in the offline mode. An exclusive (X) lock on the target table is acquired before any rows are inserted. This prevents concurrent applications from accessing table data. This is the default import behavior.

ALLOW WRITE ACCESS
Runs import in the online mode. An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the **REPLACE**, **CREATE**, or **REPLACE_CREATE** import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the **COMMITCOUNT** option was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required

when you import to a nickname and **COMMITCOUNT** must be specified with a valid number (AUTOMATIC is not considered a valid option).

AS ROOT TABLE
Creates one or more sub-tables as a stand-alone table hierarchy.

COMMITCOUNT *n* **| AUTOMATIC**
Performs a COMMIT after every *n* records are imported. When a number *n* is specified, import performs a COMMIT after every *n* records are imported. When compound inserts are used, a user-specified commit frequency of *n* is rounded up to the first integer multiple of the compound count value. When AUTOMATIC is specified, import internally determines when a commit needs to be performed. The utility will commit for either one of two reasons:

- to avoid running out of active log space
- to avoid lock escalation from row level to table level

If the **ALLOW WRITE ACCESS** option is specified, and the **COMMITCOUNT** option is not specified, the import utility will perform commits as if **COMMITCOUNT** AUTOMATIC had been specified.

The ability of the import operation to avoid running out of active log space is affected by the DB2 registry variable **DB2_FORCE_APP_ON_MAX_LOG**:

- If **DB2_FORCE_APP_ON_MAX_LOG** is set to FALSE and the **COMMITCOUNT** AUTOMATIC command option is specified, the import utility will be able to automatically avoid running out of active log space.
- If **DB2_FORCE_APP_ON_MAX_LOG** is set to FALSE and the **COMMITCOUNT** *n* command option is specified, the import utility will attempt to resolve the log full condition if it encounters an SQL0964C (Transaction Log Full) while inserting or updating a record. It will perform an unconditional commit and then will reattempt to insert or update the record. If this does not help resolve the issue (which would be the case when the log full is attributed to other activity on the database), then the IMPORT command will fail as expected, however the number of rows committed may not be a multiple of the **COMMITCOUNT** *n* value. To avoid processing the rows that were already committed when you retry the import operation, use the **RESTARTCOUNT** or **SKIPCOUNT** command parameters.
- If **DB2_FORCE_APP_ON_MAX_LOG** is set to TRUE (which is the default), the import operation will fail if it encounters an SQL0964C while inserting or updating a record. This can occur irrespective of whether you specify **COMMITCOUNT** AUTOMATIC or **COMMITCOUNT** *n*.

  The application is forced off the database and the current unit of work is rolled back. To avoid processing the rows that were already committed when you retry the import operation, use the **RESTARTCOUNT** or **SKIPCOUNT** command parameters.

CREATE

Note: The **CREATE** parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options **CREATE** and **REPLACE_CREATE** are deprecated".

Creates the table definition and row contents in the code page of the database. If the data was exported from a DB2 table, sub-table, or hierarchy, indexes are created. If this option operates on a hierarchy, and data was exported from DB2, a type hierarchy will also be created. This option can only be used with IXF files.

This parameter is not valid when you import to a nickname.

Note: If the data was exported from an MVS™ host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are more than 254, **CREATE** might fail because the rows are too long. See "Imported table re-creation" for a list of restrictions. In this case, the table should be created manually, and IMPORT with **INSERT** should be invoked, or, alternatively, the LOAD command should be used.

DEFAULT *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The schema specified through the **DEFAULT** clause identifies a schema to use for validation when the XML Data Specifier (XDS) of an imported XML document does not contain an SCH attribute identifying an XML Schema.

The **DEFAULT** clause takes precedence over the **IGNORE** and **MAP** clauses. If an XDS satisfies the **DEFAULT** clause, the **IGNORE** and **MAP** specifications will be ignored.

FROM *filename*
Specifies the name of the file that contains the data to be imported. This must be a fully qualified path and the file must exist on the database server.

HIERARCHY
Specifies that hierarchical data is to be imported.

IGNORE *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The **IGNORE** clause specifies a list of one or more schemas to ignore if they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for an imported XML document, and the schema identified by the SCH attribute is included in the list of schemas to ignore, then no schema validation will occur for the imported XML document.

If a schema is specified in the **IGNORE** clause, it cannot also be present in the left side of a schema pair in the **MAP** clause.

The **IGNORE** clause applies only to the XDS. A schema that is mapped by the **MAP** clause will not be subsequently ignored if specified by the **IGNORE** clause.

IN *tablespace-name*
Identifies the table space in which the table will be created. The table space must exist, and must be a REGULAR table space. If no other table space is specified, all table parts are stored in this table space. If this clause is not specified, the table is created in a table space created by the authorization ID. If none is found, the table is placed into the default table space USERSPACE1. If USERSPACE1 has been dropped, table creation fails.

INDEX IN *tablespace-name*
Identifies the table space in which any indexes on the table will be created. This option is allowed only when the primary table space specified in the

IN clause is a DMS table space. The specified table space must exist, and must be a REGULAR or LARGE DMS table space.

Note: Specifying which table space will contain an index can only be done when the table is created.

insert-column
Specifies the name of a column in the table or the view into which data is to be inserted.

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds rows of imported data to the target table, or updates existing rows (of the target table) with matching primary keys.

INTO *table-name*
Specifies the database table into which the data is to be imported. This table cannot be a system table, a created temporary table, a declared temporary table, or a summary table.

One can use an alias for **INSERT**, **INSERT_UPDATE**, or **REPLACE**, except in the case of an earlier server, when the fully qualified or the unqualified table name should be used. A qualified table name is in the form: *schema.tablename*. The *schema* is the user name under which the table was created.

LOBS FROM *lob-path*
Specifies one or more fully qualified paths that store LOB files. The paths must exist on the database server coordinator partition. The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

This parameter is not valid when you import to a nickname.

LONG IN *tablespace-name*
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, or distinct types with any of these as source types) will be stored. This option is allowed only if the primary table space specified in the **IN** clause is a DMS table space. The table space must exist, and must be a LARGE DMS table space.

MAP *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. Use the **MAP** clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each imported XML document. The **MAP** clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the **MAP** clause, it cannot also be specified in the **IGNORE** clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

MESSAGES ON SERVER
Specifies that the message file created on the server by the IMPORT command is to be saved. The result set returned will include the following two columns: MSG_RETRIEVAL, which is the SQL statement required to retrieve all the warnings and error messages that occur during this operation, and MSG_REMOVAL, which is the SQL statement required to clean up the messages.

If this clause is not specified, the message file will be deleted when the ADMIN_CMD procedure returns to the caller. The MSG_RETRIEVAL and MSG_REMOVAL column in the result set will contain null values.

Note that with or without the clause, the fenced user ID must have the authority to create files under the directory indicated by the **DB2_UTIL_MSGPATH** registry variable, as well as the directory where the data is to be exported to.

METHOD

L Specifies the start and end column numbers from which to import data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1.

Note: This method can only be used with ASC files, and is the only valid option for that file type.

N Specifies the names of the columns in the data file to be imported. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the **METHOD N** list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, method N (F2, F1, F4, F3) is a valid request, while method N (F2, F1) is not valid.

Note: This method can only be used with IXF files.

P Specifies the field numbers of the input data fields to be imported.

Note: This method can only be used with IXF or DEL files, and is the only valid option for the DEL file type.

MODIFIED BY *filetype-mod*
Specifies file type modifier options. See "File type modifiers for the import utility" on page 94.

NOTIMEOUT
Specifies that the import utility will not time out while waiting for locks. This option supersedes the **locktimeout** database configuration parameter. Other applications are not affected.

NULL INDICATORS *null-indicator-list*
This option can only be used when the **METHOD L** parameter is specified. That is, the input file is an ASC file. The null indicator list is a

comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the **METHOD L** parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the **METHOD L** option will be imported.

The NULL indicator character can be changed using the **MODIFIED BY** option, with the `nullindchar` file type modifier.

OF *filetype*

Specifies the format of the data in the input file:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony
- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2.

Important: Support for the WSF file format is deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed.

The WSF file type is not supported when you import to a nickname.

REPLACE

Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed. This option can only be used if the table exists. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

This option does not honor the CREATE TABLE statement's NOT LOGGED INITIALLY (NLI) clause or the ALTER TABLE statement's ACTIVE NOT LOGGED INITIALLY clause.

If an import with the **REPLACE** option is performed within the same transaction as a CREATE TABLE or ALTER TABLE statement where the NLI clause is invoked, the import will not honor the NLI clause. All inserts will be logged.

Workaround 1

Delete the contents of the table using the DELETE statement, then invoke the import with INSERT statement

Workaround 2

Drop the table and recreate it, then invoke the import with INSERT statement.

This limitation applies to DB2 Universal Database Version 7 and DB2 UDB Version 8

REPLACE_CREATE

Note: The **REPLACE_CREATE** parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

If the table exists, deletes all existing data from the table by truncating the data object, and inserts the imported data without changing the table definition or the index definitions.

If the table does not exist, creates the table and index definitions, as well as the row contents, in the code page of the database. See *Imported table re-creation* for a list of restrictions.

This option can only be used with IXF files. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

RESTARTCOUNT *n*
Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to **SKIPCOUNT**. **RESTARTCOUNT** and **SKIPCOUNT** are mutually exclusive.

ROWCOUNT *n*
Specifies the number *n* of physical records in the file to be imported (inserted or updated). Allows a user to import only *n* rows from a file, starting from the record determined by the **SKIPCOUNT** or **RESTARTCOUNT** options. If the **SKIPCOUNT** or **RESTARTCOUNT** options are not specified, the first *n* rows are imported. If **SKIPCOUNT** *m* or **RESTARTCOUNT** *m* is specified, rows *m*+1 to *m*+*n* are imported. When compound inserts are used, user specified **ROWCOUNT** *n* is rounded up to the first integer multiple of the compound count value.

SKIPCOUNT *n*
Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to **RESTARTCOUNT**. **SKIPCOUNT** and **RESTARTCOUNT** are mutually exclusive.

STARTING *sub-table-name*
A keyword for hierarchy only, requesting the default order, starting from *sub-table-name*. For PC/IXF files, the default order is the order stored in the input file. The default order is the only valid order for the PC/IXF file format.

sub-table-list
For typed tables with the **INSERT** or the **INSERT_UPDATE** option, a list of sub-table names is used to indicate the sub-tables into which data is to be imported.

traversal-order-list
For typed tables with the **INSERT**, **INSERT_UPDATE**, or the **REPLACE** option, a list of sub-table names is used to indicate the traversal order of the importing sub-tables in the hierarchy.

UNDER *sub-table-name*
Specifies a parent table for creating one or more sub-tables.

WARNINGCOUNT *n*
Stops the import operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail. If *n* is zero, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

XML FROM *xml-path*
Specifies one or more paths that contain the XML files.

XMLPARSE
Specifies how XML documents are parsed. If this option is not specified, the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE
Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE
Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the **XMLVALIDATE** option is invoked with the **USING XDS** clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the **DEFAULT** clause.

The **DEFAULT**, **IGNORE**, and **MAP** clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the **DEFAULT** clause, it will not be ignored if also specified by the **IGNORE** clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the MAP clause, it will not be re-mapped if also specified in the second part of another **MAP** clause pair.

USING SCHEMA *schema-sqlid*
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the **USING SCHEMALOCATION HINTS** clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the **XMLVALIDATE** option below.

Example

The following example shows how to import information from the file `myfile.ixf` to the STAFF table in the SAMPLE database.

```
CALL SYSPROC.ADMIN_CMD
   ('IMPORT FROM /home/userid/data/myfile.ixf
   OF IXF MESSAGES ON SERVER INSERT INTO STAFF')
```

Usage notes

Any path used in the IMPORT command must be a valid fully-qualified path on the coordinator node for the server.

If the **ALLOW WRITE ACCESS** or **COMMITCOUNT** options are specified, a commit will be performed by the import utility. This causes the ADMIN_CMD procedure to return an SQL30090N error with reason code 1 in the case of Type 2 connections.

If the value to be assigned for a column of a result set from the ADMIN_CMD procedure is greater than the maximum value for the data type of the column, then the maximum value for the data type is assigned and a warning message, SQL1155W, is returned.

Be sure to complete all table operations and release all locks before starting an import operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

The import utility adds rows to the target table using the SQL INSERT statement. The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.
- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a **REPLACE** or a **REPLACE_CREATE** operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a **CREATE**, **REPLACE**, or **REPLACE_CREATE** operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the **REPLACE** or the **REPLACE_CREATE** option to rerun the whole import operation, or use **INSERT** with the **RESTARTCOUNT** parameter set to the number of rows successfully imported.

Updates from the IMPORT command will always be committed at the end of an IMPORT task. The IMPORT command can also perform automatic commits during

its execution to reduce the size of the lock list and the active log space. The IMPORT command will rollback if the active log becomes full during IMPORT processing.

- By default, automatic commits are not performed for the **INSERT** or the **INSERT_UPDATE** option. They are, however, performed if the **COMMITCOUNT** parameter is not zero.
- Offline import does not perform automatic COMMITs if any of the following conditions are true:
  - The target is a view, not a table
  - Compound inserts are used
  - Buffered inserts are used
- By default, online import performs automatic commit to free both the active log space and the lock list. Automatic commits are not performed only if a **COMMITCOUNT** value of zero is specified.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

You cannot **REPLACE** or **REPLACE_CREATE** an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:

1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when recreating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using `SELECT *`.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60 KB), the message file returned to the user on the client might be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

Importing PC/IXF files to a remote database is much faster if the PC/IXF file is on a hard drive rather than on diskettes.

The database table or hierarchy must exist before data in the **ASC**, **DEL**, or **WSF** file formats can be imported; however, if the table does not already exist, IMPORT **CREATE** or IMPORT **REPLACE_CREATE** creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT **CREATE** can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand. The file copying step is not necessary if the source and the target databases are both accessible from the same client.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the **FORCEIN** option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the **FORCEIN** option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the **FORCEIN** option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 clients on the AIX operating system.

For table objects on an 8 KB page that are close to the limit of 1012 columns, import of PC/IXF data files might cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of **DEL** or **ASC** files. If PC/IXF files are being used to create a new table, an alternative is use db2look to dump the DDL statement that created the table, and then to issue that statement through the CLP.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (**INSERT** option) is supported. The **RESTARTCOUNT** parameter, but not the **COMMITCOUNT** parameter, is also supported.

When using the **CREATE** option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than **CREATE** with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file.

The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a created temporary table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

Importing a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. Only the name of the first file

must be specified in the IMPORT command. For example, `IMPORT FROM data.ixf OF IXF INSERT INTO TABLE1`. The file `data.002`, etc should be available in the same directory as `data.ixf`.

On the Windows operating system:

- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Security labels in their internal format might contain newline characters. If you import the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the `delprioritychar` file type modifier in the IMPORT command.

Federated considerations

When using the IMPORT command and the **INSERT**, **UPDATE**, or **INSERT_UPDATE** command parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you want to use when doing an import operation already exists. There are also several restrictions you should be aware of as shown in the IMPORT command parameters section.

Some data sources, such as ODBC, do not support importing into nicknames.

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the command returns additional information in result sets as follows:

Table 42. Result set returned by the IMPORT command

Column name	Data type	Description
ROWS_READ	BIGINT	Number of records read from the file during import.
ROWS_SKIPPED	BIGINT	Number of records skipped before inserting or updating begins.
ROWS_INSERTED	BIGINT	Number of rows inserted into the target table.
ROWS_UPDATED	BIGINT	Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).
ROWS_REJECTED	BIGINT	Number of records that could not be imported.
ROWS_COMMITTED	BIGINT	Number of records imported successfully and committed to the database.

MSG_RETRIEVAL	VARCHAR(512)	SQL statement that is used to retrieve messages created by this utility. For example:
		`SELECT SQLCODE, MSG`
		`FROM TABLE (SYSPROC.ADMIN_GET_MSGS`
		`('1203498_txu')) AS MSG`

MSG_REMOVAL	VARCHAR(512)	SQL statement that is used to clean up messages created by this utility. For example:
		`CALL SYSPROC.ADMIN_REMOVE_MSGS`
		`('1203498_txu')`

File type modifiers for the import utility

Table 43. Valid file type modifiers for the import utility: All file formats

Modifier	Description
compound=*x*	*x* is a number between 1 and 100 inclusive. Uses nonatomic compound SQL to insert the data, and *x* statements will be attempted each time. If this modifier is specified, and the transaction log is not sufficiently large, the import operation will fail. The transaction log must be large enough to accommodate either the number of rows specified by **COMMITCOUNT**, or the number of rows in the data file if **COMMITCOUNT** is not specified. It is therefore recommended that the **COMMITCOUNT** option be specified to avoid transaction log overflow. This modifier is incompatible with **INSERT_UPDATE** mode, hierarchical tables, and the following modifiers: `usedefaults`, `identitymissing`, `identityignore`, `generatedmissing`, and `generatedignore`.
generatedignore	This modifier informs the import utility that data for all generated columns is present in the data file but should be ignored. This results in all values for the generated columns being generated by the utility. This modifier cannot be used with the `generatedmissing` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated columns (not even NULLs), and will therefore generate a value for each row. This modifier cannot be used with the `generatedignore` modifier.
identityignore	This modifier informs the import utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `identitymissing` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with the `identityignore` modifier.
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *`filename.ext`* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and `mmm` is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file `db2exp.001`, and is 456 bytes long. The **LOBS FROM** clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The **LOBS FROM** clause will implicitly activate the LOBSINFILE behavior. The **LOBS FROM** clause conveys to the IMPORT utility the list of paths to search for the LOB files while importing the data. To indicate a null LOB, enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
no_type_id	Valid only when importing into a single sub-table. Typical usage is to export data from a regular table, and then to invoke an import operation (using this modifier) to convert the data into a single sub-table.

Table 43. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
nodefaults	If a source column for a target table column is not explicitly specified, and the table column is not nullable, default values are not loaded. Without this option, if a source column for one of the target table columns is not explicitly specified, one of the following occurs: • If a default value can be specified for a column, the default value is loaded • If the column is nullable, and a default value cannot be specified for that column, a NULL is loaded • If the column is not nullable, and a default value cannot be specified, an error is returned, and the utility stops processing.
norowwarnings	Suppresses all warnings about rejected rows.
rowchangetimestampignore	This modifier informs the import utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMP being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `rowchangetimestampmissing` modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with the `rowchangetimestampignore` modifier.
seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. IMPORT converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W)) is returned. This modifier cannot be specified if the seclabelname modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned.
seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. IMPORT will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned. This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned. **Note:** If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.

Table 43. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. **Note:** For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 44. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data from this code page to the application code page during the import operation. The following rules apply: • For pure DBCS (graphic) mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. **Note:** 1. The `codepage` modifier cannot be used with the `lobsinfile` modifier. 2. If data expansion occurs when the code page is converted from the application code page to the database code page, the data might be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[2] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

Table 44. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.

timeformat="*x*"	*x* is the format of the time in the source file.[2] Valid time elements are:
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`TT - Meridian indicator (AM or PM)`
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 44. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[2] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`U (1 to 12 times) - Fractional seconds(number of occurrences of U represent the number of digits with each digit ranging from 0 to 9`
	`TT - Meridian indicator (AM or PM)`
	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	The following example illustrates how to import data containing user defined date and time formats into a table called schedule:
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

Table 44. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being imported into graphic or double-byte character large object (DBCLOB) data fields is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic code page is associated with the character code page. IMPORT determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the application if the `codepage` modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The `usegraphiccodepage` modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The `usegraphiccodepage` modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the `codepage` file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute. **Note:** If the `xmlgraphic` modifier is specified with the IMPORT command, the XML document to be imported must be encoded in the UTF-16 code page. Otherwise, the XML document may be rejected with a parsing error, or it may be imported into the table with data corruption.

Table 45. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format

Modifier	Description
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Table 45. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format (continued)

Modifier	Description
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[3] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the null indicator character is specified to be the letter `N`, then `n` is also recognized as a null indicator.
reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.

striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.
	
	In the following example, `striptblanks` causes the import utility to truncate trailing blank spaces:
	`db2 import from myfile.asc of asc`
	`  modified by striptblanks`
	`  method l (1 10, 12 15) messages msgs.txt`
	`  insert into staff`
	
	This option cannot be specified together with `striptnulls`. These are mutually exclusive options. This option replaces the obsolete `t` option, which is supported for earlier compatibility only.

striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. This option replaces the obsolete `padwithzero` option, which is supported for earlier compatibility only.

Table 46. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[34] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	
	The single quotation mark (') can also be specified as a character string delimiter. In the following example, `chardel''` causes the import utility to interpret any single quotation mark (') it encounters as a character string delimiter:
	`db2 "import from myfile.del of del`
	`  modified by chardel''`
	`  method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[34]
	
	In the following example, `coldel;` causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 import from myfile.del of del`
	`  modified by coldel;`
	`  messages msgs.txt insert into staff`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.

Table 46. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[34]
	
	In the following example, `decpt;` causes the import utility to interpret any semicolon (;) it encounters as a decimal point:
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`decpt; messages msgs.txt insert into staff"`

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 import ... modified by delprioritychar ...`
	
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
nochardel	The import utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx`, `delprioritychar` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 47. Valid file type modifiers for the import utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to import each row.
indexixf	Directs the utility to drop all indexes currently defined on the existing table, and to create new ones from the index definitions in the PC/IXF file. This option can only be used when the contents of a table are being replaced. It cannot be used with a view, or when a *insert-column* is specified.

Table 47. Valid file type modifiers for the import utility: IXF file format (continued)

Modifier	Description
indexschema=*schema*	Uses the specified *schema* for the index name during index creation. If *schema* is not specified (but the keyword `indexschema` *is* specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
forcecreate	Specifies that the table should be created with possible missing or limited information after returning SQL3311N during an import operation.

Table 48. IMPORT behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	IMPORT behavior
Absent	Absent	All data in the file is assumed to be in the application code page.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the application code page. Graphic data is assumed to be in the code page of the application graphic data. If the application code page is single-byte, then all data is assumed to be in the application code page. **Warning:** If the application code page is single-byte, graphic data will be corrupted when imported into the database, even if the database contains graphic columns.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.

Note:

1. The import utility does not issue a warning if an attempt is made to use unsupported file types with the **MODIFIED BY** option. If this is attempted, the import operation fails, and an error code is returned.
2. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end

positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

```
"M" (could be a month, or a minute)
"M:M" (Which is which?)
"M:YYYY:M" (Both are interpreted as month.)
"S:M:YYYY" (adjacent to both a time value and a date value)
```

In ambiguous cases, the utility will report an error message, and the operation will fail.

Following are some unambiguous time stamp formats:

```
"M:YYYY" (Month)
"S:M" (Minute)
"M:YYYY:S:M" (Month....Minute)
"M:H:YYYY:M:D" (Minute....Month)
```

Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).

3. Character values provided for the chardel, coldel, or decpt file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```

4. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
5. The following file type modifiers are not allowed when importing into a nickname:
   - `indexixf`
   - `indexschema`
   - `dldelfiletype`
   - `nodefaults`
   - `usedefaults`
   - `no_type_idfiletype`
   - `generatedignore`
   - `generatedmissing`
   - `identityignore`
   - `identitymissing`
   - `lobsinfile`
6. The WSF file format is not supported for XML columns. Support for this file format is also deprecated and might be removed in a future release. It is recommended that you start using a supported file format instead of WSF files before support is removed
7. The **CREATE** mode is not supported for XML columns.

8. All XML data must reside in XML files that are separate from the main data file. An XML Data Specifier (XDS) (or a NULL value) must exist for each XML column in the main data file.
9. XML documents are assumed to be in Unicode format or to contain a declaration tag that includes an encoding attribute, unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. Rows containing documents that are not well-formed will be rejected.
11. If the **XMLVALIDATE** option is specified, documents that successfully validate against their matching schema will be annotated with the schema information as they are inserted. Rows containing documents that fail to validate against their matching schema will be rejected. To successfully perform the validation, the privileges held by the user invoking the import must include at least one of the following:
    - DBADM authority
    - USAGE privilege on the XML schema to be used in the validation
12. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the rowchangetimestampmissing file type modifier *must be* specified in the import command if data for the column is not present in the data to be imported and there is no explicit column list present.

INITIALIZE TAPE command using the ADMIN_CMD procedure

Initializes tapes for backup and restore operations to streaming tape devices. This command is only supported on Windows operating systems.

Authorization

One of the following:
- SYSADM
- SYSCTRL
- SYSMAINT

Required connection

Database

Command syntax

Command parameters

ON *device*
Specifies a valid tape device name. The default value is `\\.\TAPE0`. The device specified must be relative to the server.

USING *blksize*
Specifies the block size for the device, in bytes. The device is initialized to use the block size specified, if the value is within the supported range of block sizes for the device.

The buffer size specified for the BACKUP DATABASE command and for RESTORE DATABASE must be divisible by the block size specified here.

If a value for this parameter is not specified, the device is initialized to use its default block size. If a value of zero is specified, the device is initialized to use a variable length block size; if the device does not support variable length block mode, an error is returned.

When backing up to tape, use of a variable block size is currently not supported. If you must use this option, ensure that you have well tested procedures in place that enable you to recover successfully, using backup images that were created with a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Example

Initialize the tape device to use a block size of 2048 bytes, if the value is within the supported range of block sizes for the device.

```
CALL SYSPROC.ADMIN_CMD( 'initialize tape using 2048' )
```

Usage notes

Command execution status is returned in the SQLCA resulting from the CALL statement.

LOAD command using the ADMIN_CMD procedure

Loads data into a DB2 table.

Data residing on the server can be in the form of a file, tape, or named pipe. Data can also be loaded from a cursor defined from a query running against the currently connected database or a different database under the same instance, or by using a user-written script or application. If the COMPRESS attribute for the table is set to YES, the data loaded will be subject to compression on every data and database partition for which a dictionary already exists in the table, including data in the XML storage object of the table.

Quick link to “File type modifiers for the load utility” on page 130.

Restrictions

The load utility does not support loading data at the hierarchy level. The load utility is not compatible with range-clustered tables. The load utility does not support the NOT LOGGED INITIALLY parameter for the CREATE TABLE or ALTER TABLE statements.

Scope

This command can be issued against multiple database partitions in a single request.

Authorization

One of the following:

- DATAACCESS
- LOAD authority on the database and

  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the load fails and an error (SQLSTATE 5U014) is returned.
- To load data into a table that has protected rows, the session authorization id must hold a security label that meets these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access or for all access

  If the session authorization id does not hold such a security label then the load fails and an error (SQLSTATE 5U014) is returned. This security label is used to protect a loaded row if the session authorization ID's LBAC credentials do not allow it to write to the security label that protects that row in the data. This does not happen, however, when the security policy protecting the table was created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option of the CREATE SECURITY POLICY statement. In this case the load fails and an error (SQLSTATE 42519) is returned.
- If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.
- If the **LOCK WITH FORCE** option is specified, SYSADM authority is required.

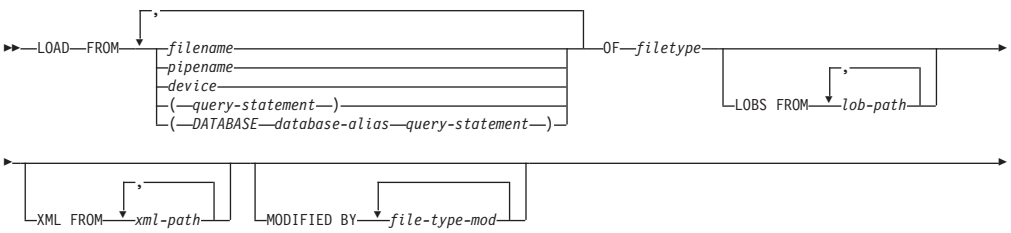
Since all load processes (and all DB2 server processes, in general) are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

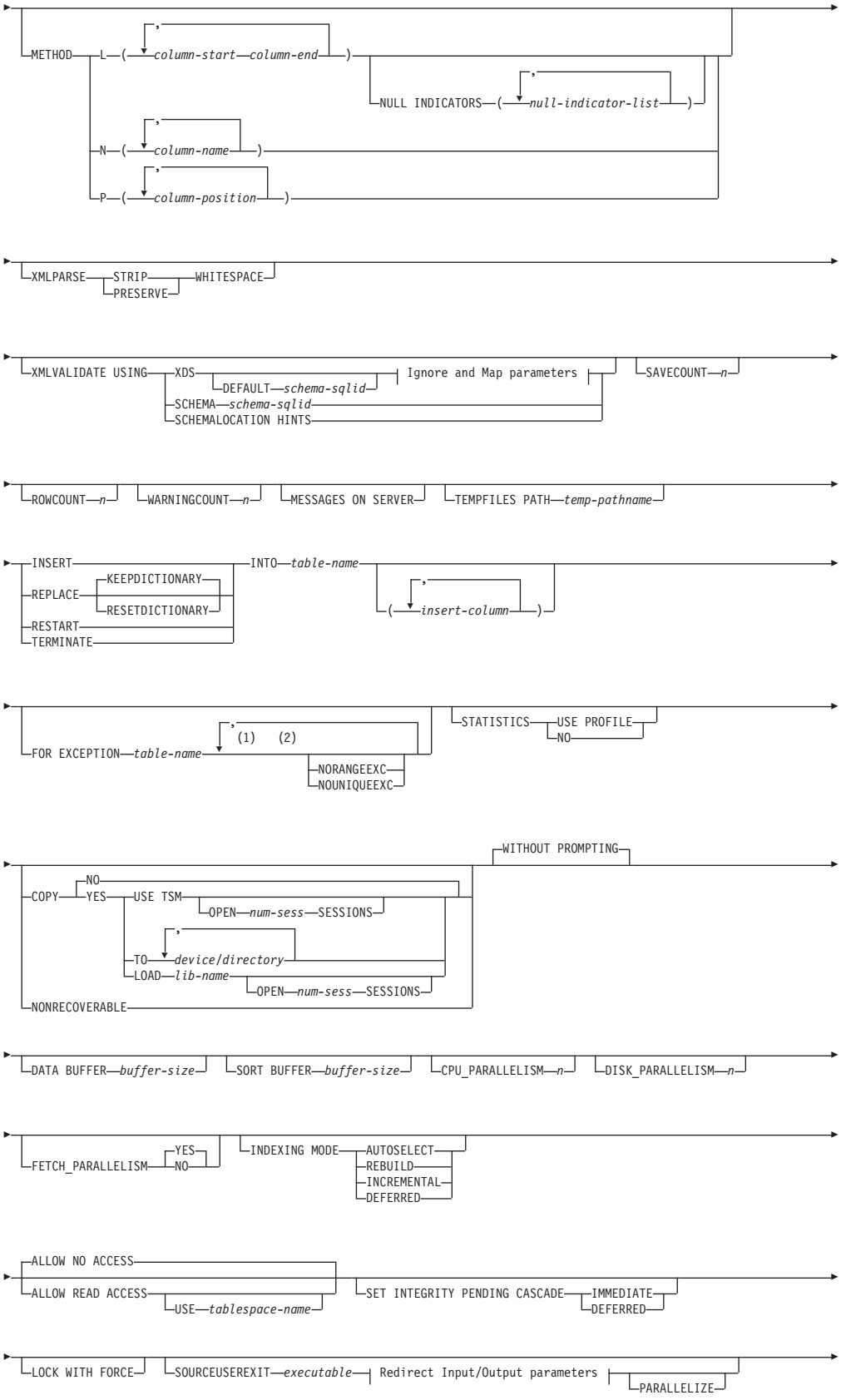
Required connection

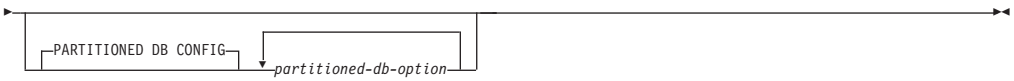
Database.

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

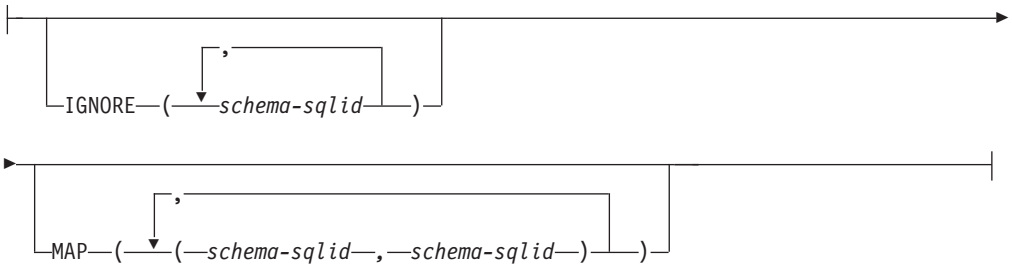
Command syntax



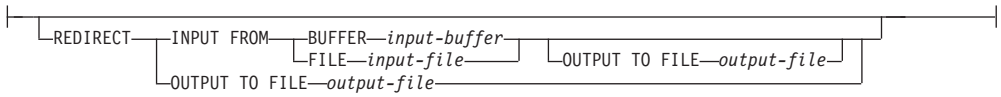

METHOD
L
column-start
column-end
NULL INDICATORS
null-indicator-list
N
column-name
P
column-position
XMLPARSE
STRIP
PRESERVE
WHITESPACE
XMLVALIDATE USING
XDS
DEFAULT
schema-sqlid
Ignore and Map parameters
SCHEMA
SCHEMALOCATION HINTS
SAVECOUNT
n
ROWCOUNT
WARNINGCOUNT
MESSAGES ON SERVER
TEMPFILES PATH
temp-pathname
INSERT
REPLACE
KEEPDICTIONARY
RESETDICTIONARY
RESTART
TERMINATE
INTO
table-name
insert-column
FOR EXCEPTION
(1) (2)
NORANGEEXC
NOUNIQUEEXC
STATISTICS
USE PROFILE
NO
WITHOUT PROMPTING
COPY
NO
YES
USE TSM
OPEN
num-sess
SESSIONS
TO
device/directory
LOAD
lib-name
NONRECOVERABLE
DATA BUFFER
buffer-size
SORT BUFFER
CPU_PARALLELISM
DISK_PARALLELISM
FETCH_PARALLELISM
INDEXING MODE
AUTOSELECT
REBUILD
INCREMENTAL
DEFERRED
ALLOW NO ACCESS
ALLOW READ ACCESS
USE
tablespace-name
SET INTEGRITY PENDING CASCADE
IMMEDIATE
DEFERRED
LOCK WITH FORCE
SOURCEUSEREXIT
executable
Redirect Input/Output parameters
PARALLELIZE



Ignore and Map parameters:



Redirect Input/Output parameters:



Notes:

1 These keywords can appear in any order.

2 Each of these keywords can only appear once.

Command parameters

FROM *filename* | *pipename* | *device(query-statement)* | **(DATABASE** *database-alias query-statement***)**

Specifies the file, pipe or device referring to an SQL statement that contains the data being loaded, or the SQL statement itself and the optional source database to load from cursor.

The *query-statement* option is used to LOAD from a cursor. It contains only one query statement, which is enclosed in parentheses, and can start with VALUES, SELECT or WITH. For example,

```
LOAD FROM (SELECT * FROM T1) OF CURSOR INSERT INTO T2
```

When the **DATABASE** *database-alias* clause is included prior to the query statement in the parentheses, the LOAD command will attempt to load the data using the *query-statement* from the given database as indicated by the *database-alias* name, which is defined on the server. It must point to a database exist on the server, and is a different database that the application is currently connected to. Note that the LOAD will be executed using the user ID and password explicitly provided for the currently connected database (an implicit connection will cause the LOAD to fail).

If the input source is a file, pipe, or device, it must be accessible from the coordinator partition on the server.

If several names are specified, they will be processed in sequence. If the last item specified is a tape device and the user is prompted for a tape, the LOAD will fail and the ADMIN_CMD procedure will return an error.

Note:

1. A fully qualified path file name must be used and must exist on the server.
2. If data is exported into a file using the EXPORT command using the ADMIN_CMD procedure, the data file is owned by the fenced user ID. This file is not usually accessible by the instance owner. To run the LOAD from CLP or the ADMIN_CMD procedure, the data file must be accessible by the instance owner ID, so read access to the data file must be granted to the instance owner.
3. Loading data from multiple IXF files is supported if the files are physically separate, but logically one file. It is *not* supported if the files are both logically and physically separate. (Multiple physical files would be considered logically one if they were all created with one invocation of the EXPORT command.)
4. When loading XML data from files into tables in a partitioned database environment, the XML data files must be read-accessible to all the database partitions where loading is taking place.

OF *filetype*

Specifies the format of the data:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format)
- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2 databases.
- CURSOR (a cursor declared against a SELECT or VALUES statement).

Note: When using a CURSOR file type to load XML data into a table in a distributed database environment, the PARTITION_ONLY and LOAD_ONLY modes are not supported.

LOBS FROM *lob-path*

The path to the data files containing LOB values to be loaded. The path must end with a slash. The path must be fully qualified and accessible from the coordinator partition on the server . The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the **LOBSINFILE** behavior.

This option is ignored when specified in conjunction with the CURSOR file type.

MODIFIED BY *file-type-mod*

Specifies file type modifier options. See "File type modifiers for the load utility" on page 130.

METHOD

L Specifies the start and end column numbers from which to load data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1. This method can only be used with ASC files, and is the only valid method for that file type.

NULL INDICATORS *null-indicator-list*

This option can only be used when the **METHOD L** parameter is specified; that is, the input file is an ASC file). The null indicator list is a comma-separated list of positive integers specifying the column number of each null

indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the **METHOD L** parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the **METHOD L** option will be loaded.

The NULL indicator character can be changed using the **MODIFIED BY** option.

N Specifies the names of the columns in the data file to be loaded. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the **METHOD N** list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method N (F2, F1, F4, F3)` is a valid request, while `method N (F2, F1)` is not valid. This method can only be used with file types IXF or CURSOR.

P Specifies the field numbers (numbered from 1) of the input data fields to be loaded. Each table column that is not nullable should have a corresponding entry in the **METHOD P** list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method P (2, 1, 4, 3)` is a valid request, while `method P (2, 1)` is not valid. This method can only be used with file types IXF, DEL, or CURSOR, and is the only valid method for the DEL file type.

XML FROM *xml-path*
Specifies one or more paths that contain the XML files. XDSs are contained in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the XML column.

XMLPARSE
Specifies how XML documents are parsed. If this option is not specified, the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE
Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE
Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the **XMLVALIDATE** option is invoked with the **USING XDS** clause, the schema used to perform validation will be determined

by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the **DEFAULT** clause.

The **DEFAULT**, **IGNORE**, and **MAP** clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the **DEFAULT** clause, it will not be ignored if also specified by the **IGNORE** clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the **MAP** clause, it will not be re-mapped if also specified in the second part of another **MAP** clause pair.

USING SCHEMA *schema-sqlid*
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the **USING SCHEMALOCATION HINTS** clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the **XMLVALIDATE** option below.

IGNORE *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The **IGNORE** clause specifies a list of one or more schemas to ignore if they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for a loaded XML document, and the schema identified by the SCH attribute is included in the list of schemas to ignore, then no schema validation will occur for the loaded XML document.

Note:

If a schema is specified in the **IGNORE** clause, it cannot also be present in the left side of a schema pair in the **MAP** clause.

The **IGNORE** clause applies only to the XDS. A schema that is mapped by the **MAP** clause will not be subsequently ignored if specified by the **IGNORE** clause.

DEFAULT *schema-sqlid*
This option can only be used when the **USING XDS** parameter is specified. The schema specified through the **DEFAULT** clause identifies a schema to use for validation when the XML Data Specifier (XDS) of a loaded XML document does not contain an SCH attribute identifying an XML Schema.

The **DEFAULT** clause takes precedence over the **IGNORE** and **MAP** clauses. If an XDS satisfies the **DEFAULT** clause, the **IGNORE** and **MAP** specifications will be ignored.

MAP *schema-sqlid*
This option can only be used when the **USING XDS** parameter is

specified. Use the **MAP** clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each loaded XML document. The **MAP** clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the **MAP** clause, it cannot also be specified in the **IGNORE** clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

SAVECOUNT *n*
Specifies that the load utility is to establish consistency points after every *n* rows. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using LOAD QUERY. If the value of *n* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is zero, meaning that no consistency points will be established, unless necessary.

This option is ignored when specified in conjunction with the CURSOR file type or when loading a table containing an XML column.

ROWCOUNT *n*
Specifies the number of *n* physical records in the file to be loaded. Allows a user to load only the first *n* rows in a file.

WARNINGCOUNT *n*
Stops the load operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If *n* is zero, or this option is not specified, the load operation will continue regardless of the number of warnings issued. If the load operation is stopped because the threshold of warnings was encountered, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

MESSAGES ON SERVER
Specifies that the message file created on the server by the LOAD command is to be saved. The result set returned will include the following two columns: MSG_RETRIEVAL, which is the SQL statement required to retrieve all the warnings and error messages that occur during this operation, and MSG_REMOVAL, which is the SQL statement required to clean up the messages.

If this clause is not specified, the message file will be deleted when the ADMIN_CMD procedure returns to the caller. The MSG_RETRIEVAL and MSG_REMOVAL column in the result set will contain null values.

Note that with or without the clause, the fenced user ID must have the authority to create files under the directory indicated by the **DB2_UTIL_MSGPATH** registry variable.

TEMPFILES PATH *temp-pathname*
Specifies the name of the path to be used when creating temporary files during a load operation, and should be fully qualified according to the server database partition.

Temporary files take up file system space. Sometimes, this space requirement is quite substantial. Following is an estimate of how much file system space should be allocated for all temporary files:

- 136 bytes for each message that the load utility generates
- 15 KB overhead if the data file contains long field data or LOBs. This quantity can grow significantly if the **INSERT** option is specified, and there is a large amount of long field or LOB data already in the table.

INSERT
One of four modes under which the load utility can execute. Adds the loaded data to the table without changing the existing table data.

REPLACE
One of four modes under which the load utility can execute. Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

KEEPDICTIONARY
An existing compression dictionary is preserved across the LOAD REPLACE operation. Provided the table COMPRESS attribute is YES, the newly replaced data is subject to being compressed using the dictionary that existed prior to the invocation of the load. If no dictionary previously existed in the table, a new dictionary is built using the data that is being replaced into the table as long as the table COMPRESS attribute is YES. The amount of data that is required to build the compression dictionary in this case is subject to the policies of ADC. This data is populated into the table as uncompressed. Once the dictionary is inserted into the table, the remaining data to be loaded is subject to being compressed with this dictionary. This is the default parameter. For summary, see Table 1 below.

The following example keeps the old dictionary if it is currently in the table:

```
CALL SYSPROC.ADMIN_CMD('load from staff.del of del replace
 keepdictionary into SAMPLE.STAFF statistics use profile
 data buffer 8')
```

Table 49. LOAD REPLACE KEEPDICTIONARY

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
YES	YES	YES	Preserve table row data and XML dictionaries.	Data to be loaded is subject to compression.

Table 49. LOAD REPLACE KEEPDICTIONARY (continued)

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
YES	YES	NO	Preserve table row data dictionary and build a new XML dictionary.	Table row data to be loaded is subject to compression. After XML dictionary is built, remaining XML data to be loaded is subject to compression.
YES	NO	YES	Build table row data dictionary and preserve XML dictionary.	After table row data dictionary is built, remaining table row data to be loaded is subject to compression. XML data to be loaded is subject to compression.
YES	NO	NO	Build new table row data and XML dictionaries.	After dictionaries are built, remaining data to be loaded is subject to compression.
NO	YES	YES	Preserve table row data and XML dictionaries.	Data to be loaded is not compressed.
NO	YES	NO	Preserve table row data dictionary.	Data to be loaded is not compressed.
NO	NO	YES	No effect on table row dictionary. Preserve XML dictionary.	Data to be loaded is not compressed.
NO	NO	NO	No effect.	Data to be loaded is not compressed.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.

RESETDICTIONARY

This directive instructs LOAD REPLACE processing to build a new dictionary for the table data object provided that the table COMPRESS attribute is YES. If the COMPRESS attribute is NO and a dictionary was already present in the table it will be removed and no new dictionary will be inserted into the table. A compression dictionary can be built with just one user record. If the loaded data set size is zero and if there is a preexisting dictionary, the dictionary will not be preserved. The amount of data required to build a dictionary with this directive is not subject to the policies of ADC. For summary, see Table 2 below.

The following example will reset the current dictionary and make a new one:

```
CALL SYSPROC.ADMIN_CMD('load from staff.del of del replace
 resetdictionary into SAMPLE.STAFF statistics use profile
 data buffer 8')
```

Table 50. LOAD REPLACE RESETDICTIONARY

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
YES	YES	YES	Build new dictionaries[2]. If the DATA CAPTURE CHANGES option is enabled on the CREATE TABLE or ALTER TABLE statements, the current table row data dictionary is kept (and referred to as the *historical compression dictionary*).	After dictionaries are built, remaining data to be loaded is subject to compression.
YES	YES	NO	Build new dictionaries[2]. If the DATA CAPTURE CHANGES option is enabled on the CREATE TABLE or ALTER TABLE statements, the current table row data dictionary is kept (and referred to as the *historical compression dictionary*).	After dictionaries are built, remaining data to be loaded is subject to compression.
YES	NO	YES	Build new dictionaries.	After dictionaries are built, remaining data to be loaded is subject to compression.
YES	NO	NO	Build new dictionaries.	After dictionaries are built, remaining data to be loaded is subject to compression.
NO	YES	YES	Remove dictionaries.	Data to be loaded is not compressed.
NO	YES	NO	Remove table row data dictionary.	Data to be loaded is not compressed.
NO	NO	YES	Remove XML storage object dictionary.	Data to be loaded is not compressed.
NO	NO	NO	No effect.	All table data is not compressed.

Notes:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.
2. If a dictionary exists and the compression attribute is enabled, but there are no records to load into the table partition, a new dictionary cannot be built and the **RESETDICTIONARY** operation will not keep the existing dictionary.

TERMINATE

One of four modes under which the load utility can execute. Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects might be marked as

invalid, in which case index rebuild will automatically take place at next access). If the load operation being terminated is a LOAD REPLACE, the table will be truncated to an empty table after the LOAD TERMINATE operation. If the load operation being terminated is a LOAD INSERT, the table will retain all of its original records after the LOAD TERMINATE operation. For summary of dictionary management, see Table 3 below.

The LOAD **TERMINATE** option will not remove a backup pending state from table spaces.

RESTART
One of four modes under which the load utility can execute. Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase. For summary of dictionary management, see Table 4 below.

INTO *table-name*
Specifies the database table into which the data is to be loaded. This table cannot be a system table, a declared temporary table, or a created temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form *schema.tablename*. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

insert-column
Specifies the table column into which the data is to be inserted.

The load utility cannot parse columns whose names contain one or more spaces. For example,

```
CALL SYSPROC.ADMIN_CMD('load from delfile1 of del noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, Int 4, I5, I6, DT7, I8, TM9)')
```

will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

```
CALL SYSPROC.ADMIN_CMD('load from delfile1 of del noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, "Int 4", I5, I6, DT7, I8, TM9)')
```

FOR EXCEPTION *table-name*
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

Information that is written to the exception table is *not* written to the dump file. In a partitioned database environment, an exception table must be defined for those database partitions on which the loading table is defined. The dump file, otherwise, contains rows that cannot be loaded because they are invalid or have syntax errors.

When loading XML data, using the **FOR EXCEPTION** clause to specify a load exception table is not supported in the following cases:

- When using label-based access control (LBAC).
- When loading data into a partitioned table.

NORANGEEXC
Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

NOUNIQUEEXC
Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

STATISTICS USE PROFILE
Instructs load to collect statistics during the load according to the profile defined for this table. This profile must be created before load is executed. The profile is created by the RUNSTATS command. If the profile does not exist and load is instructed to collect statistics according to the profile, a warning is returned and no statistics are collected.

During load, distribution statistics are not collected for columns of type XML.

STATISTICS NO
Specifies that no statistics are to be collected, and that the statistics in the catalogs are not to be altered. This is the default.

COPY NO
Specifies that the table space in which the table resides will be placed in backup pending state if forward recovery is enabled (that is, **logretain** or **userexit** is on). The **COPY NO** option will also put the table space state into the Load in Progress table space state. This is a transient state that will disappear when the load completes or aborts. The data in any table in the table space cannot be updated or deleted until a table space backup or a full database backup is made. However, it is possible to access the data in any table by using the SELECT statement.

LOAD with **COPY NO** on a recoverable database leaves the table spaces in a backup pending state. For example, performing a LOAD with **COPY NO** and **INDEXING MODE DEFERRED** will leave indexes needing a refresh. Certain queries on the table might require an index scan and will not succeed until the indexes are refreshed. The index cannot be refreshed if it resides in a table space which is in the backup pending state. In that case, access to the table will not be allowed until a backup is taken. Index refresh is done automatically by the database when the index is accessed by a query. If one of **COPY NO**, **COPY YES**, or **NONRECOVERABLE** is not specified, and the database is recoverable (**logretain** or **logarchmeth1** is enabled), then **COPY NO** is the default.

COPY YES
Specifies that a copy of the loaded data will be saved. This option is invalid if forward recovery is disabled.

USE TSM
Specifies that the copy will be stored using Tivoli Storage Manager (TSM).

OPEN *num-sess* **SESSIONS**
The number of I/O sessions to be used with TSM or the vendor product. The default value is 1.

TO *device/directory*
Specifies the device or directory on which the copy image will be created.

LOAD *lib-name*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path where the user exit programs reside.

NONRECOVERABLE
Specifies that the load transaction is to be marked as nonrecoverable and that it will not be possible to recover it by a subsequent roll forward action. The roll forward utility will skip the transaction and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward operation is completed, such a table can only be dropped or restored from a backup (full or table space) taken after a commit point following the completion of the non-recoverable load operation.

With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. If one of **COPY NO**, **COPY YES**, or **NONRECOVERABLE** is not specified, and the database is not recoverable (**logretain** or **logarchmeth1** is not enabled), then **NONRECOVERABLE** is the default.

WITHOUT PROMPTING
Specifies that the list of data files contains all the files that are to be loaded, and that the devices or directories listed are sufficient for the entire load operation. If a continuation input file is not found, or the copy targets are filled before the load operation finishes, the load operation will fail, and the table will remain in load pending state.

This is the default. Any actions which normally require user intervention will return an error message.

DATA BUFFER *buffer-size*
Specifies the number of 4 KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the minimum required resource is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the **util_heap_sz** database configuration parameter. Beginning in version 9.5, the value of the DATA BUFFER option of the LOAD command can temporarily exceed **util_heap_sz** if more memory is available in the system. In this situation, the utility heap is dynamically increased as needed until the **database_memory** limit is reached. This memory will be released once the load operation completes.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

SORT BUFFER *buffer-size*
This option specifies a value that overrides the **sortheap** database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the **INDEXING MODE** parameter is not specified as DEFERRED. The value that is specified cannot exceed the value of **sortheap**. This parameter is useful for throttling the sort memory that is used when loading tables with many indexes without changing the value of **sortheap**, which would also affect general query processing.

CPU_PARALLELISM *n*
Specifies the number of processes or threads that the load utility will create for parsing, converting, and formatting records when building table objects. This parameter is designed to exploit the number of processes

running per database partition. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, or has not been specified, the load utility uses an intelligent default value (usually based on the number of CPUs available) at run time.

Note:

1. If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs or the value specified by the user.
2. Specifying a small value for the **SAVECOUNT** parameter causes the loader to perform many more I/O operations to flush both data and table metadata. When **CPU_PARALLELISM** is greater than one, the flushing operations are asynchronous, permitting the loader to exploit the CPU. When **CPU_PARALLELISM** is set to one, the loader waits on I/O during consistency points. A load operation with **CPU_PARALLELISM** set to two, and **SAVECOUNT** set to 10 000, completes faster than the same operation with **CPU_PARALLELISM** set to one, even though there is only one CPU.

DISK_PARALLELISM *n*

Specifies the number of processes or threads that the load utility will create for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

FETCH_PARALLELISM YES | NO

When performing a load from a cursor where the cursor is declared using the **DATABASE** keyword, or when using the API sqlu_remotefetch_entry media entry, and this option is set to YES, the load utility attempts to parallelize fetching from the remote data source if possible. If set to NO, no parallel fetching is performed. The default value is YES. For more information, see "Moving data using the CURSOR file type".

INDEXING MODE

Specifies whether the load utility is to rebuild indexes or to extend them incrementally. Valid values are:

AUTOSELECT

The load utility will automatically decide between REBUILD or INCREMENTAL mode. The decision is based on the amount of data being loaded and the depth of the index tree. Information relating to the depth of the index tree is stored in the index object. RUNSTATS is not required to populate this information. AUTOSELECT is the default indexing mode.

REBUILD

All indexes will be rebuilt. The utility must have sufficient resources to sort all index key parts for both old and appended table data.

INCREMENTAL

Indexes will be extended with new data. This approach consumes index free space. It only requires enough sort space to append index keys for the inserted records. This method is only supported in cases where the index object is valid and accessible at the start of a load operation (it is, for example, not valid immediately following a load operation in which the DEFERRED mode was

specified). If this mode is specified, but not supported due to the state of the index, a warning is returned, and the load operation continues in REBUILD mode. Similarly, if a load restart operation is begun in the load build phase, INCREMENTAL mode is not supported.

DEFERRED
The load utility will not attempt index creation if this mode is specified. Indexes will be marked as needing a refresh. The first access to such indexes that is unrelated to a load operation might force a rebuild, or indexes might be rebuilt when the database is restarted. This approach requires enough sort space for all key parts for the largest index. The total time subsequently taken for index construction is longer than that required in REBUILD mode. Therefore, when performing multiple load operations with deferred indexing, it is advisable (from a performance viewpoint) to let the last load operation in the sequence perform an index rebuild, rather than allow indexes to be rebuilt at first non-load access.

Deferred indexing is only supported for tables with non-unique indexes, so that duplicate keys inserted during the load phase are not persistent after the load operation.

ALLOW NO ACCESS
Load will lock the target table for exclusive access during the load. The table state will be set to Load In Progress during the load. **ALLOW NO ACCESS** is the default behavior. It is the only valid option for LOAD REPLACE.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending state.

ALLOW READ ACCESS
Load will lock the target table in a share mode. The table state will be set to both Load In Progress and Read Access. Readers can access the non-delta portion of the data while the table is being load. In other words, data that existed before the start of the load will be accessible by readers to the table, data that is being loaded is not available until the load is complete. LOAD TERMINATE or LOAD RESTART of an **ALLOW READ ACCESS** load can use this option; LOAD TERMINATE or LOAD RESTART of an **ALLOW NO ACCESS** load cannot use this option. Furthermore, this option is not valid if the indexes on the target table are marked as requiring a rebuild.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress, and Read Access. At the end of the load, the table state Load In Progress will be removed but the table states Set Integrity Pending and Read Access will remain. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending. While the table is in Set Integrity Pending and Read Access states, the non-delta portion of the data is still accessible to readers, the new (delta) portion of the data will remain inaccessible until the SET INTEGRITY statement has completed. A user can perform multiple loads on the same table without issuing a SET INTEGRITY statement. Only the original (checked) data will remain visible, however, until the SET INTEGRITY statement is issued.

ALLOW READ ACCESS also supports the following modifiers:

USE *tablespace-name*

If the indexes are being rebuilt, a shadow copy of the index is built in table space *tablespace-name* and copied over to the original table space at the end of the load during an INDEX COPY PHASE. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same table space as the index object. If the shadow copy is created in the same table space as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different table space from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load during the INDEX COPY PHASE.

Without this option the shadow index is built in the same table space as the original. Since both the original index and shadow index by default reside in the same table space simultaneously, there might be insufficient space to hold both indexes within one table space. Using this option ensures that you retain enough table space for the indexes.

This option is ignored if the user does not specify **INDEXING MODE REBUILD** or **INDEXING MODE AUTOSELECT**. This option will also be ignored if **INDEXING MODE AUTOSELECT** is chosen and load chooses to incrementally update the index.

SET INTEGRITY PENDING CASCADE

If LOAD puts the table into Set Integrity Pending state, the **SET INTEGRITY PENDING CASCADE** option allows the user to specify whether or not Set Integrity Pending state of the loaded table is immediately cascaded to all descendents (including descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables).

IMMEDIATE

Indicates that Set Integrity Pending state is immediately extended to all descendent foreign key tables, descendent immediate materialized query tables and descendent staging tables. For a LOAD INSERT operation, Set Integrity Pending state is not extended to descendent foreign key tables even if the **IMMEDIATE** option is specified.

When the loaded table is later checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement), descendent foreign key tables that were placed in Set Integrity Pending Read Access state will be put into Set Integrity Pending No Access state.

DEFERRED

Indicates that only the loaded table will be placed in the Set Integrity Pending state. The states of the descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged.

Descendent foreign key tables might later be implicitly placed in Set Integrity Pending state when their parent tables are checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement). Descendent immediate materialized query tables and descendent immediate staging tables

will be implicitly placed in Set Integrity Pending state when one of its underlying tables is checked for integrity violations. A query of a table that is in the Set Integrity Pending state might succeed if an eligible materialized query table that is not in the Set Integrity Pending state is accessed by the query instead of the specified table. A warning (SQLSTATE 01586) will be issued to indicate that descendent tables have been placed in Set Integrity Pending state. See the Notes® section of the SET INTEGRITY statement in the SQL Reference for when these descendent tables will be put into Set Integrity Pending state.

If the SET INTEGRITY PENDING CASCADE option is not specified:

- Only the loaded table will be placed in Set Integrity Pending state. The state of descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged, and can later be implicitly put into Set Integrity Pending state when the loaded table is checked for constraint violations.

If LOAD does not put the target table into Set Integrity Pending state, the **SET INTEGRITY PENDING CASCADE** option is ignored.

LOCK WITH FORCE
The utility acquires various locks including table locks in the process of loading. Rather than wait, and possibly timeout, when acquiring a lock, this option allows load to force off other applications that hold conflicting locks on the target table. Applications holding conflicting locks on the system catalog tables will not be forced off by the load utility. Forced applications will roll back and release the locks the load utility needs. The load utility can then proceed. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

ALLOW NO ACCESS loads might force applications holding conflicting locks at the start of the load operation. At the start of the load the utility can force applications that are attempting to either query or modify the table.

ALLOW READ ACCESS loads can force applications holding conflicting locks at the start or end of the load operation. At the start of the load the load utility can force applications that are attempting to modify the table. At the end of the load operation, the load utility can force applications that are attempting to either query or modify the table.

SOURCEUSEREXIT *executable*
Specifies an executable filename which will be called to feed data into the utility.

REDIRECT

INPUT FROM

BUFFER *input-buffer*
The stream of bytes specified in *input-buffer* is passed into the STDIN file descriptor of the process executing the given executable.

FILE *input-file*
The contents of this client-side file are passed into the STDIN file descriptor of the process executing the given executable.

OUTPUT TO

FILE *output-file*
The STDOUT and STDERR file descriptors are captured to the fully qualified server-side file specified.

PARALLELIZE
Increases the throughput of data coming into the load utility by invoking multiple user exit processes simultaneously. This option is only applicable in multi-partition database environments and is ignored in single-partition database environments.

For more information, see "Moving data using a customized application (user exit)".

PARTITIONED DB CONFIG *partitioned-db-option*
Allows you to execute a load into a table distributed across multiple database partitions. The **PARTITIONED DB CONFIG** parameter allows you to specify partitioned database-specific configuration options. The *partitioned-db-option* values can be any of the following:

```
PART_FILE_LOCATION x
OUTPUT_DBPARTNUMS x
PARTITIONING_DBPARTNUMS x
MODE x
MAX_NUM_PART_AGENTS x
ISOLATE_PART_ERRS x
STATUS_INTERVAL x
PORT_RANGE x
CHECK_TRUNCATION
MAP_FILE_INPUT x
MAP_FILE_OUTPUT x
TRACE x
NEWLINE
DISTFILE x
OMIT_HEADER
RUN_STAT_DBPARTNUM x
```

Detailed descriptions of these options are provided in "Load configuration options for partitioned database environments".

RESTARTCOUNT
Reserved.

USING *directory*
Reserved.

Example

Issue a load with replace option for the employee table data from a file.

```
CALL SYSPROC.ADMIN_CMD('LOAD FROM /home/theresax/tmp/emp_exp.dat
   OF DEL METHOD P (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)
   MESSAGES /home/theresax/tmp/emp_load.msg
   REPLACE INTO THERESAX.EMPLOYEE (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY,
   BONUS, COMM) COPY NO INDEXING MODE AUTOSELECT ISOLATE_PART_ERRS
   LOAD_ERRS_ONLY MODE PARTITION_AND_LOAD' )
```

The following is an example of output from a single-partition database.

```
Result set 1
--------------

ROWS_READ     ROWS_SKIPPED     ROWS_LOADED     ROWS_REJECTED     ...
---------...- ------------...- -----------...- -------------...- ...
```

```
          32               0             32               0 ...

  1 record(s) selected.

Return Status = 0
```

Output from a single-partition database (continued).

```
... ROWS_DELETED     ROWS_COMMITTED     MSG_RETRIEVAL
... ------------...- --------------...- ------------------------------...-
...               0                32 SELECT SQLCODE, MSG_TEXT FROM
...                                      TABLE(SYSPROC.ADMIN_GET_MSGS(
...                                      '2203498_thx')) AS MSG
```

Output from a single-partition database (continued).

```
... MSG_REMOVAL
... ----------------------------------------...-
... CALL SYSPROC.ADMIN_REMOVE_MSGS('2203498_thx')
...
```

Note: The following columns are also returned in this result set, but are set to NULL because they are only populated when loading into a multi-partition database: ROWS_PARTITIONED and NUM_AGENTINFO_ENTRIES.

The following is an example of output from a multi-partition database.

```
Result set 1
--------------

ROWS_READ     ROWS_REJECTED     ROWS_PARTITIONED     NUM_AGENTINFO_ENTRIES ...
---------...- -------------...- -----------------...- ---------------------- ...
          32                 0                   32                      5 ...
                                                                           ...
                                                                           ...


  1 record(s) selected.
```

Output from a multi-partition database (continued).

```
... MSG_RETRIEVAL                          MSG_REMOVAL
... ---------------------------------...- ----------------------------...-
... SELECT DBPARTITIONNUM, AGENT_TYPE,     CALL SYSPROC.ADMIN_REMOVE_MSGS
...   SQLCODE, MSG_TEXT FROM TABLE             ('2203498_thx')
...   (SYSPROC.ADMIN_GET_MSGS
...   ('2203498_thx')) AS MSG
```

Note: The following columns are also returned in this result set, but are set to NULL because they are only populated when loading into a single-partition database: ROWS_SKIPPED, ROWS_LOADED, ROWS_DELETED and ROWS_COMMITTED.

Output from a multi-partition database (continued).

```
Result set 2
--------------

DBPARTITIONNUM     SQLCODE     TABSTATE     AGENTTYPE
--------------...- -------...- --------...- ------------...-
               10           0 NORMAL       LOAD
               20           0 NORMAL       LOAD
               30           0 NORMAL       LOAD
               20           0 NORMAL       PARTITION
               10           0 NORMAL       PRE_PARTITION
```

```
1 record(s) selected.

Return Status = 0
```

Examples of loading data from XML documents

Loading XML data

Example 1

The user has constructed a data file with XDS fields to describe the documents that are to be inserted into the table. It might appear like this :

```
1, "<XDS FIL=""file1.xml"" />"
2, "<XDS FIL='file2.xml' OFF='23' LEN='45' />"
```

For the first row, the XML document is identified by the file named `file1.xml`. Note that since the character delimiter is the double quote character, and double quotation marks exist inside the XDS, the double quotation marks contained within the XDS are doubled. For the second row, the XML document is identified by the file named `file2.xml`, and starts at byte offset 23, and is 45 bytes in length.

Example 2

The user issues a load command without any parsing or validation options for the XML column, and the data is loaded successfully:

```
LOAD
FROM data.del of DEL INSERT INTO mytable
```

Loading XML data from CURSOR

Loading data from cursor is the same as with a regular relational column type. The user has two tables, T1 and T2, each of which consist of a single XML column named C1. To LOAD from T1 into T2, the user will first declare a cursor:

```
DECLARE
X1 CURSOR FOR SELECT C1 FROM T1;
```

Next, the user may issue a LOAD using the cursor type:

```
LOAD FROM X1 of
CURSOR INSERT INTO T2
```

Applying the XML specific LOAD options to the cursor type is the same as loading from a file.

Usage notes

- Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted. If preservation of the source data order is not required, consider using the **ANYORDER** file type modifier, described below in the “File type modifiers for the load utility” section.
- The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update materialized query tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in Set Integrity Pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are

dependent on tables being loaded, are also placed in Set Integrity Pending state. Issue the SET INTEGRITY statement to take the tables out of Set Integrity Pending state. Load operations cannot be carried out on replicated materialized query tables.

- If a clustering index exists on the table, the data should be sorted on the clustering index prior to loading. Data does not need to be sorted prior to loading into a multidimensional clustering (MDC) table, however.
- If you specify an exception table when loading into a protected table, any rows that are protected by invalid security labels will be sent to that table. This might allow users that have access to the exception table to access to data that they would not normally be authorized to access. For better security be careful who you grant exception table access to, delete each row as soon as it is repaired and copied to the table being loaded, and drop the exception table as soon as you are done with it.
- Security labels in their internal format might contain newline characters. If you load the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the **delprioritychar** file type modifier in the LOAD command.
- The LOAD utility issues a COMMIT statement at the beginning of the operation which, in the case of Type 2 connections, causes the procedure to return SQL30090N with reason code 1.
- Any path used in the LOAD command must be a valid fully-qualified path on the server coordinator partition.
- For performing a load using the CURSOR file type where the DATABASE keyword was specified during the DECLARE CURSOR statement, the user ID and password used to authenticate against the database currently connected to (for the load) will be used to authenticate against the source database (specified by the DATABASE option of the DECLARE CURSOR statement). If no user ID or password was specified for the connection to the loading database, a user ID and password for the source database must be specified during the DECLARE CURSOR statement.
- Loading a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. The names of all the files must be specified in the LOAD command. For example, `LOAD FROM DATA.IXF, DATA.002 OF IXF INSERT INTO TABLE1`. Loading to the Windows operating system from logically split PC/IXF files is not supported.
- When restarting a failed LOAD, the behavior will follow the existing behavior in that the BUILD phase will be forced to use the REBUILD mode for indexes.
- Loading XML documents between databases is not supported and returns error message SQL1407N.
- The LOAD utility does not support loading into tables that contain columns that reference fenced procedures. If you issue the LOAD command on such table, you will receive error message SQL1376N. To work around this restriction, you can redefine the routine to be unfenced, or use the import utility.

Summary of LOAD TERMINATE and LOAD RESTART dictionary management

The following chart summarizes the compression dictionary management behavior for LOAD processing under the **TERMINATE** directive.

Table 51. LOAD TERMINATE dictionary management

Table COMPRESS attribute	Does table row data dictionary exist prior to LOAD?	XML storage object dictionary exists prior to LOAD[1]	TERMINATE: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	TERMINATE: LOAD REPLACE RESETDICTIONARY
YES	YES	YES	Keep existing dictionaries.	Neither dictionary is kept. [2]
YES	YES	NO	Keep existing dictionary.	Nothing is kept. [2]
YES	NO	YES	Keep existing dictionary.	Nothing is kept.
YES	NO	NO	Nothing is kept.	Nothing is kept.
NO	YES	YES	Keep existing dictionaries.	Nothing is kept.
NO	YES	NO	Keep existing dictionary.	Nothing is kept.
NO	NO	YES	Keep existing dictionary.	Nothing is kept.
NO	NO	NO	Do nothing.	Do nothing.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.
2. In the special case that the table has data capture enabled, the table row data dictionary is kept.

LOAD RESTART truncates a table up to the last consistency point reached. As part of LOAD RESTART processing, a compression dictionary will exist in the table if it was present in the table at the time the last LOAD consistency point was taken. In that case, LOAD RESTART will not create a new dictionary. For a summary of the possible conditions, see Table 4 below.

Table 52. LOAD RESTART dictionary management

Table COMPRESS Attribute	Table row data dictionary exist prior to LOAD consistency point?[1]	XML Storage object dictionary existed prior to last LOAD?[2]	RESTART: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	RESTART: LOAD REPLACE RESETDICTIONARY
YES	YES	YES	Keep existing dictionaries.	Keep existing dictionaries.
YES	YES	NO	Keep existing table row data dictionary and build XML dictionary subject to ADC.	Keep existing table row data dictionary and build XML dictionary.
YES	NO	YES	Build table row data dictionary subject to ADC. Keep existing XML dictionary.	Build table row data dictionary. Keep existing XML dictionary.
YES	NO	NO	Build table row data and XML dictionaries subject to ADC.	Build table row data and XML dictionaries.
NO	YES	YES	Keep existing dictionaries.	Remove existing dictionaries.
NO	YES	NO	Keep existing table row data dictionary.	Remove existing table row data dictionary.

Table 52. LOAD RESTART dictionary management (continued)

Table COMPRESS Attribute	Table row data dictionary exist prior to LOAD consistency point?[1]	XML Storage object dictionary existed prior to last LOAD?[2]	RESTART: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	RESTART: LOAD REPLACE RESETDICTIONARY
NO	NO	YES	Keep existing XML dictionary.	Remove existing XML dictionary.
NO	NO	NO	Do nothing.	Do nothing.

Notes:

1. The **SAVECOUNT** option is ignored when loading XML data, load operations that fail during the load phase restart from the beginning of the operation.
2. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated using an online table move.

Result set information

Command execution status is returned in the SQLCA resulting from the CALL statement. If execution is successful, the command returns additional information. A single-partition database will return one result set; a multi-partition database will return two result sets.

- Table 53: Result set for a load operation.
- Table 54 on page 129: Result set 2 contains information for each database partition in a multi-partition load operation.

Table 53. Result set returned by the LOAD command

Column name	Data type	Description
ROWS_READ	BIGINT	Number of rows read during the load operation.
ROWS_SKIPPED	BIGINT	Number of rows skipped before the load operation started. This information is returned for a single-partition database only.
ROWS_LOADED	BIGINT	Number of rows loaded into the target table. This information is returned for a single-partition database only.
ROWS_REJECTED	BIGINT	Number of rows that could not be loaded into the target table.
ROWS_DELETED	BIGINT	Number of duplicate rows that were not loaded into the target table. This information is returned for a single-partition database only.
ROWS_COMMITTED	BIGINT	Total number of rows processed: the number of rows successfully loaded into the target table, plus the number of skipped and rejected rows. This information is returned for a single-partition database only.
ROWS_PARTITIONED	BIGINT	Number of rows distributed by all database distributing agents. This information is returned for a multi-partition database only.

Table 53. Result set returned by the LOAD command (continued)

Column name	Data type	Description
NUM_AGENTINFO_ENTRIES	BIGINT	Number of entries returned in the second result set for a multi-partition database. This is the number of agent information entries produced by the load operation. This information is returned for multi-partition database only.

MSG_RETRIEVAL	VARCHAR(512)	SQL statement that is used to retrieve messages created by this utility. For example,
		`SELECT SQLCODE, MSG`
		`FROM TABLE`
		`(SYSPROC.ADMIN_GET_MSGS`
		`('2203498_thx')) AS MSG`
		
		This information is returned only if the **MESSAGES ON SERVER** clause is specified.

MSG_REMOVAL	VARCHAR(512)	SQL statement that is used to clean up messages created by this utility. For example:
		`CALL SYSPROC.ADMIN_REMOVE_MSGS`
		`('2203498_thx')`
		
		This information is returned only if the **MESSAGES ON SERVER** clause is specified.

Table 54. Result set 2 returned by the LOAD command for each database partition in a multi-partition database.

Column name	Data type	Description
DBPARTITIONNUM	SMALLINT	The database partition number on which the agent executed the load operation.
SQLCODE	INTEGER	Final SQLCODE resulting from the load processing.

Table 54. Result set 2 returned by the LOAD command for each database partition in a multi-partition database. (continued)

Column name	Data type	Description
TABSTATE	VARCHAR(20)	Table state after load operation has completed. It is one of: • LOADPENDING: Indicates that the load did not complete, but the table on the partition has been left in a LOAD PENDING state. A load restart or terminate operation must be done on the database partition. • NORMAL: Indicates that the load completed successfully on the database partition and the table was taken out of the LOAD IN PROGRESS (or LOAD PENDING) state. Note that the table might still be in Set Integrity Pending state if further constraints processing is required, but this state is not reported by this interface. • UNCHANGED: Indicates that the load did not complete due to an error, but the state of the table has not yet been changed. It is not necessary to perform a load restart or terminate operation on the database partition. **Note:** Not all possible table states are returned by this interface.
AGENTTYPE	VARCHAR(20)	Agent type and is one of: • FILE_TRANSFER • LOAD • LOAD_TO_FILE • PARTITIONING • PRE_PARTITIONING

File type modifiers for the load utility

Table 55. Valid file type modifiers for the load utility: All file formats

Modifier	Description
anyorder	This modifier is used in conjunction with the **cpu_parallelism** parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of **cpu_parallelism** is 1, this option is ignored. This option is not supported if **SAVECOUNT** > 0, since crash recovery after a consistency point requires that data be loaded in sequence.
generatedignore	This modifier informs the load utility that data for all generated columns is present in the data file but should be ignored. This results in all generated column values being generated by the utility. This modifier cannot be used with either the **generatedmissing** or the **generatedoverride** modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated column (not even NULLs). This results in all generated column values being generated by the utility. This modifier cannot be used with either the **generatedignore** or the **generatedoverride** modifier.

Table 55. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description

generatedoverride	This modifier instructs the load utility to accept user-supplied data for all generated columns in the table (contrary to the normal rules for these types of columns). This is useful when migrating data from another database system, or when loading a table from data that was recovered using the **RECOVER DROPPED TABLE** option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for a non-nullable generated column will be rejected (SQL3116W). When this modifier is used, the table will be placed in Set Integrity Pending state. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR table-name GENERATED COLUMN IMMEDIATE UNCHECKED`
	
	To take the table out of Set Integrity Pending state and force verification of the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR table-name IMMEDIATE CHECKED.`
	
	When this modifier is specified and there is a generated column in any of the partitioning keys, dimension keys or distribution keys, then the LOAD command will automatically convert the modifier to **generatedignore** and proceed with the load. This will have the effect of regenerating all of the generated column values.
	
	This modifier cannot be used with either the **generatedmissing** or the **generatedignore** modifier.

identityignore	This modifier informs the load utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the **identitymissing** or the **identityoverride** modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with either the **identityignore** or the **identityoverride** modifier.
identityoverride	This modifier should be used only when an identity column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of identity columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the **DROPPED TABLE RECOVERY** option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the identity column will be rejected (SQL3116W). This modifier cannot be used with either the **identitymissing** or the **identityignore** modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's identity column when this option is used.
indexfreespace=*x*	*x* is an integer between 0 and 99 inclusive. The value is interpreted as the percentage of each index page that is to be left as free space when load rebuilds the index. Load with **INDEXING MODE INCREMENTAL** ignores this option. The first entry in a page is added without restriction; subsequent entries are added to maintain the percent free space threshold. The default value is the one used at CREATE INDEX time. This value takes precedence over the PCTFREE value specified in the CREATE INDEX statement. The **indexfreespace** option affects index leaf pages only.

Table 55. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column. This option is not supported in conjunction with the CURSOR filetype. The **LOBS FROM** clause specifies where the LOB files are located when the **lobsinfile** modifier is used. The **LOBS FROM** clause will implicitly activate the **lobsinfile** behavior. The **LOBS FROM** clause conveys to the LOAD utility the list of paths to search for the LOB files while loading the data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename*.ext.*nnn*.*mmm*/, where *filename*.ext is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string db2exp.001.123.456/ is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
noheader	Skips the header verification code (applicable only to load operations into tables that reside in a single-partition database partition group). If the default MPP load (mode PARTITION_AND_LOAD) is used against a table residing in a single-partition database partition group, the file is not expected to have a header. Thus the **noheader** modifier is not needed. If the LOAD_ONLY mode is used, the file is expected to have a header. The only circumstance in which you should need to use the **noheader** modifier is if you wanted to perform LOAD_ONLY operation using a file that does not have a header.
norowwarnings	Suppresses all warnings about rejected rows.
pagefreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of each data page that is to be left as free space. If the specified value is invalid because of the minimum row size, (for example, a row that is at least 3 000 bytes long, and an *x* value of 50), the row will be placed on a new page. If a value of 100 is specified, each row will reside on a new page. The PCTFREE value of a table determines the amount of free space designated per page. If a **pagefreespace** value on the load operation or a PCTFREE value on a table have not been set, the utility will fill up as much space as possible on each page. The value set by **pagefreespace** overrides the PCTFREE value specified for the table.
rowchangetimestampignore	This modifier informs the load utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMPs being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the **rowchangetimestampmissing** or the **rowchangetimestampoverride** modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with either the **rowchangetimestampignore** or the **rowchangetimestampoverride** modifier.

Table 55. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
rowchangetimestampoverride	This modifier should be used only when a row change timestamp column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of row change timestamp columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the **DROPPED TABLE RECOVERY** option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the ROW CHANGE TIMESTAMP column will be rejected (SQL3116W). This modifier cannot be used with either the **rowchangetimestampmissing** or the **rowchangetimestampignore** modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's row change timestamp column when this option is used.

seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. LOAD converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3242W) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W) is returned.
	
	This modifier cannot be specified if the **seclabelname** modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might look like this:
	`"CONFIDENTIAL:ALPHA:G2"`
	`"CONFIDENTIAL;SIGMA:G2"`
	`"TOP SECRET:ALPHA:G2"`
	
	To load or import this data, the **seclabelchar** file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELCHAR INSERT INTO t1`

seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. LOAD will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned.
	
	This modifier cannot be specified if the **seclabelchar** modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might consist of security label names similar to:
	`"LABEL1"`
	`"LABEL1"`
	`"LABEL2"`
	
	To load or import this data, the **seclabelname** file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELNAME INSERT INTO t1`
	
	Note: If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the **striptblanks** file type modifier to make sure the spaces are removed.

Table 55. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
totalfreespace=*x*	*x* is an integer greater than or equal to 0. The value is interpreted as the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if *x* is 20, and the table has 100 data pages after the data has been loaded, 20 additional empty pages will be appended. The total number of data pages for the table will be 120. The data pages total does not factor in the number of index pages in the table. This option does not affect the index object. If two loads are done with this option specified, the second load will not reuse the extra space appended to the end by the first load.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 56. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data (and numeric data specified in characters) from this code page to the database code page during the load operation. The following rules apply: • For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • For DEL data specified in an EBCDIC code page, the delimiters might not coincide with the shift-in and shift-out DBCS characters. • **nullindchar** must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. EBCDIC data can use the corresponding symbols, even though the code points will be different. This option is not supported in conjunction with the CURSOR filetype.

Table 56. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

dateformat=*"x"*	*x* is the format of the date in the source file.[1] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

dumpfile = *x*	*x* is the fully qualified (according to the server database partition) name of an exception file to which rejected rows are written. A maximum of 32 KB of data is written per record. Following is an example that shows how to specify a dump file: `db2 load from data of del modified by dumpfile = /u/user/filename insert into table_name` The file will be created and owned by the instance owner. To override the default file permissions, use the **dumpfileaccessall** file type modifier. **Note:** 1. In a partitioned database environment, the path should be local to the loading database partition, so that concurrently running load operations do not attempt to write to the same file. 2. The contents of the file are written to disk in an asynchronous buffered mode. In the event of a failed or an interrupted load operation, the number of records committed to disk cannot be known with certainty, and consistency cannot be guaranteed after a LOAD RESTART. The file can only be assumed to be complete for a load operation that starts and completes in a single pass. 3. If the specified file already exists, it will not be recreated, but it will be truncated.
dumpfileaccessall	Grants read access to 'OTHERS' when a dump file is created. This file type modifier is only valid when: 1. it is used in conjunction with **dumpfile** file type modifier 2. the user has SELECT privilege on the load target table 3. it is issued on a DB2 server database partition that resides on a UNIX operating system If the specified file already exists, its permissions will not be changed.
fastparse	Use with caution. Reduces syntax checking on user-supplied column values, and enhances performance. Tables are guaranteed to be architecturally correct (the utility performs sufficient data checking to prevent a segmentation violation or trap), however, the coherence of the data is not validated. Only use this option if you are certain that your data is coherent and correct. For example, if the user-supplied data contains an invalid timestamp column value of `:1>0-00-20-07.11.12.000000`, this value is inserted into the table if **fastparse** is specified, and rejected if **fastparse** is not specified.

Table 56. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`. This modifier cannot be used with the **packeddecimal** modifier.

timeformat=*"x"*	*x* is the format of the time in the source file.[1] Valid time elements are:
	`H     - Hour (one or two digits ranging from 0 - 12`
	`          for a 12 hour system, and 0 - 24`
	`          for a 24 hour system)`
	`HH    - Hour (two digits ranging from 0 - 12`
	`          for a 12 hour system, and 0 - 24`
	`          for a 24 hour system; mutually exclusive`
	`            with H)`
	`M     - Minute (one or two digits ranging`
	`          from 0 - 59)`
	`MM    - Minute (two digits ranging from 0 - 59;`
	`          mutually exclusive with M)`
	`S     - Second (one or two digits ranging`
	`          from 0 - 59)`
	`SS    - Second (two digits ranging from 0 - 59;`
	`          mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits`
	`          ranging from 00000 - 86399; mutually`
	`          exclusive with other time elements)`
	`TT    - Meridian indicator (AM or PM)`
	
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 56. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat=*"x"*	*x* is the format of the time stamp in the source file.[1] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`U (1 to 12 times) - Fractional seconds(number of occurrences of U represent the number of digits with each digit ranging from 0 to 9`
	`TT - Meridian indicator (AM or PM)`

timestampformat=*"x"* (Continued)	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	If the **timestampformat** modifier is not specified, the load utility formats the timestamp field using one of two possible formats:
	`YYYY-MM-DD-HH.MM.SS`
	`YYYY-MM-DD HH:MM:SS`
	The load utility chooses the format by looking at the separator between the DD and HH. If it is a dash '-', the load utility uses the regular dashes and dots format (YYYY-MM-DD-HH.MM.SS). If it is a blank space, then the load utility expects a colon ':' to separate the HH, MM and SS.
	In either format, if you include the microseconds field (UUUUUU), the load utility expects the dot '.' as the separator. Either YYYY-MM-DD-HH.MM.SS.UUUUUU or YYYY-MM-DD HH:MM:SS.UUUUUU are acceptable.
	The following example illustrates how to load data containing user defined date and time formats into a table called schedule:
	`db2 load from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

Table 56. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If **usegraphiccodepage** is given, the assumption is made that data being loaded into graphic or double-byte character large object (DBCLOB) data field(s) is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic codepage is associated with the character code page. LOAD determines the character code page through either the **codepage** modifier, if it is specified, or through the code page of the database if the **codepage** modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The **usegraphiccodepage** modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The **usegraphiccodepage** modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the **codepage** file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the **codepage** file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute.

Table 57. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII)

Modifier	Description
binarynumerics	Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions. This option is supported only with positional ASC, using fixed length records specified by the **reclen** option. The following rules apply: • No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT. • Data lengths must match their target column definitions. • FLOATs must be in IEEE Floating Point format. • Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running. NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
nochecklengths	If **nochecklengths** is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a NULL value to *x*. The default value of *x* is `Y`.[2] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the NULL indicator character is specified to be the letter `N`, then `n` is also recognized as a NULL indicator.

packeddecimal	Loads packed-decimal data directly, since the **binarynumerics** modifier does not include the DECIMAL field type.
	
	This option is supported only with positional ASC, using fixed length records specified by the **reclen** option.
	
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
	
	Regardless of the server platform, the byte order of binary data in the load source file is assumed to be big-endian; that is, when using this modifier on Windows operating systems, the byte order must not be reversed.
	
	This modifier cannot be used with the **implieddecimal** modifier.

reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a newline character is not used to indicate the end of the row.

Table 57. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII) (continued)

Modifier	Description
striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept. This option cannot be specified together with **striptnulls**. These are mutually exclusive options. This option replaces the obsolete **t** option, which is supported for earlier compatibility only.
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with **striptblanks**. These are mutually exclusive options. This option replaces the obsolete **padwithzero** option, which is supported for earlier compatibility only.

zoneddecimal	Loads zoned decimal data, since the **binarynumerics** modifier does not include the DECIMAL field type. This option is supported only with positional ASC, using fixed length records specified by the **reclen** option.
	
	Half-byte sign values can be one of the following:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	Supported values for digits are `0x0` to `0x9`.
	
	Supported values for zones are `0x3` and `0xF`.

Table 58. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII)

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[23] If you want to explicitly specify the double quotation mark (") as the character string delimiter, you should specify it as follows:
	`modified by chardel""`
	
	The single quotation mark (') can also be specified as a character string delimiter as follows:
	`modified by chardel''`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[23]
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[23]

Table 58. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII) (continued)

Modifier	Description

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 load ... modified by delprioritychar ...`
	
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	
	With the **delprioritychar** modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
	
	The following example illustrates how to load data into a table called TABLE1, while preserving all leading and trailing spaces in the data file:
	`db2 load from delfile3 of del`
	`  modified by keepblanks`
	`  insert into table1`

nochardel	The load utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using a DB2 database system (unless **nochardel** was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with **chardelx**, **delprioritychar** or **nodoubledel**. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 59. Valid file type modifiers for the load utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If **nochecklengths** is specified, no checking is done, and an attempt is made to load each row.
nochecklengths	If **nochecklengths** is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Note:

1. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

   For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```

   Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
2. Character values provided for the **chardel**, **coldel**, or **decpt** file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
3. "Delimiter considerations for moving data" lists restrictions that apply to the characters that can be used as delimiter overrides.
4. The load utility does not issue a warning if an attempt is made to use unsupported file types with the **MODIFIED BY** option. If this is attempted, the load operation fails, and an error code is returned.
5. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the **rowchangetimestampmissing** file type modifier *must be* specified in the IMPORT command if data for the column is not present in the data to be imported and there is no explicit column list present.

Table 60. LOAD behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Absent	All data in the file is assumed to be in the database code page, not the application code page, even if the **CLIENT** option is specified.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.

Table 60. LOAD behavior when using codepage and usegraphiccodepage (continued)

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Present	Character data in the file is assumed to be in the database code page, even if the **CLIENT** option is specified. Graphic data is assumed to be in the code page of the database graphic data, even if the **CLIENT** option is specified. If the database code page is single-byte, then all data is assumed to be in the database code page. **Warning:** Graphic data will be corrupted when loaded into a single-byte database.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.

PRUNE HISTORY/LOGFILE command using the ADMIN_CMD procedure

Used to delete entries from the recovery history file or to delete log files from the active log file path of the currently connected database partition. Deleting entries from the recovery history file might be necessary if the file becomes excessively large and the retention period is high.

In a partitioned environment, the PRUNE HISTORY command only performs on the database partition it is issued on. To prune the history on multiple partitions, you can either issue the PRUNE HISTORY command from each individual database partition, or use the db2_all prefix to run the PRUNE HISTORY command on all database partitions.

Authorization

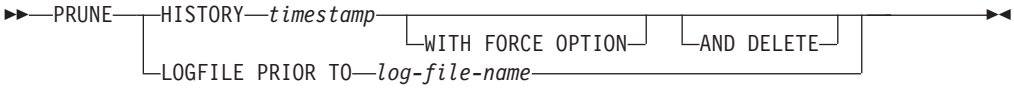
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM

Required connection

Database

Command syntax



Command parameters

HISTORY *timestamp*

Identifies a range of entries in the recovery history file that will be deleted. A complete time stamp (in the form *yyyymmddhhmmss*), or an initial prefix (minimum *yyyy*) can be specified. All entries with time stamps equal to or less than the time stamp provided are deleted from the recovery history file.

WITH FORCE OPTION

Specifies that the entries will be pruned according to the time stamp specified, even if some entries from the most recent restore set are deleted from the file. A restore set is the most recent full database backup including any restores of that backup image. If this parameter is not specified, all entries from the backup image forward will be maintained in the history.

AND DELETE

Specifies that the associated log archives will be physically deleted (based on the location information) when the history file entry is removed. This option is especially useful for ensuring that archive storage space is recovered when log archives are no longer needed. If you are archiving logs via a user exit program, the logs cannot be deleted using this option.

If you set the **auto_del_rec_obj** database configuration parameter to ON, calling PRUNE HISTORY with the **AND DELETE** parameter will also physically delete backup images and load copy images if their history file entry is pruned.

LOGFILE PRIOR TO *log-file-name*

Specifies a string for a log file name, for example `S0000100.LOG`. All log files prior to (but not including) the specified log file will be deleted. The **logretain** database configuration parameter must be set to `RECOVERY` or `CAPTURE`.

Example

Example 1: Remove all entries from the recovery history file that were written on or before December 31, 2003:

```
CALL SYSPROC.ADMIN_CMD ('prune history 20031231')
```

Example 2: Delete all log files from the active log file path prior to (but not including) `S0000100.LOG`:

```
CALL SYSPROC.ADMIN_CMD('prune logfile prior to S0000100.LOG')
```

Usage notes

If the **WITH FORCE OPTION** is used, you might delete entries that are required for automatic restoration of databases. Manual restores will still work correctly. Use of this command can also prevent the db2ckrst utility from being able to correctly analyze the complete chain of required backup images. Using the PRUNE HISTORY command without the **WITH FORCE OPTION** prevents required entries from being deleted.

Those entries with status DB2HISTORY_STATUS_DO_NOT_DELETE will not be pruned. If the **WITH FORCE OPTION** is used, then objects marked as DB2HISTORY_STATUS_DO_NOT_DELETE will still be pruned or deleted. You can set the status of recovery history file entries to

DB2HISTORY_STATUS_DO_NOT_DELETE using the UPDATE HISTORY command, the ADMIN_CMD with UPDATE_HISTORY, or the db2HistoryUpdate API. You can use the DB2HISTORY_STATUS_DO_NOT_DELETE status to prevent key recovery history file entries from being pruned and to prevent associated recovery objects from being deleted.

You can prune snapshot backup database history file entries using the PRUNE HISTORY command, but you cannot delete the related physical recovery objects using the **AND DELETE** parameter. The only way to delete snapshot backup object is to use the db2acsutil command.

The command affects only the database partition to which the application is currently connected.

QUIESCE DATABASE command using the ADMIN_CMD procedure

Forces all users off the specified instance and database and puts it into a quiesced mode.

While the database is in quiesced mode, you can perform administrative tasks on it. After administrative tasks are complete, use the UNQUIESCE command to activate the database and allow other users to connect to the database without having to shut down and perform another database start.

In this mode, only users with authority in this restricted mode are allowed to connect to the database. Users with SYSADM and DBADM authority always have access to a database while it is quiesced.

Scope

QUIESCE DATABASE results in all objects in the database being in the quiesced mode. Only the allowed user or group and SYSADM, SYSMAINT, DBADM, or SYSCTRL will be able to access the database or its objects.

Authorization

One of the following:

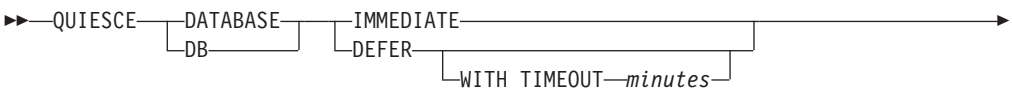
For database level quiesce:

- SYSADM
- DBADM

Required connection

Database

Command syntax



Command parameters

DEFER
Wait for applications until they commit the current unit of work.

WITH TIMEOUT *minutes*
Specifies a time, in minutes, to wait for applications to commit the current unit of work. If no value is specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the **start_stop_time** database manager configuration parameter will be used.

IMMEDIATE
Do not wait for the transactions to be committed, immediately rollback the transactions.

FORCE CONNECTIONS
Force the connections off.

DATABASE
Quiesce the database. All objects in the database will be placed in quiesced mode. Only specified users in specified groups and users with SYSADM, SYSMAINT, and SYSCTRL authority will be able to access to the database or its objects.

Example

Force off all users with connections to the database.

```
CALL SYSPROC.ADMIN_CMD( 'quiesce db immediate' )
```

- This command will force all users off the database if the **FORCE CONNECTIONS** option is supplied. **FORCE CONNECTIONS** is the default behavior; the parameter is allowed in the command for compatibility reasons.
- The command will be synchronized with the **FORCE CONNECTIONS** and will only complete once the **FORCE CONNECTIONS** has completed.

Usage notes

- After QUIESCE DATABASE, users with SYSADM, SYSMAINT, SYSCTRL, or DBADM authority, and GRANT or REVOKE privileges can designate who will be able to connect. This information will be stored permanently in the database catalog tables.

  For example,

  ```
  grant quiesce_connect on database to username/groupname
  revoke quiesce_connect on database from username/groupname
  ```

- Command execution status is returned in the SQLCA resulting from the CALL statement.

QUIESCE TABLESPACES FOR TABLE command using the ADMIN_CMD procedure

Quiesces table spaces for a table. There are three valid quiesce modes: share, intent to update, and exclusive.

There are three possible states resulting from the quiesce function:

- Quiesced: SHARE
- Quiesced: UPDATE
- Quiesced: EXCLUSIVE

Scope

In a single-partition environment, this command quiesces all table spaces involved in a load operation in exclusive mode for the duration of the load operation. In a partitioned database environment, this command acts locally on a database partition. It quiesces only that portion of table spaces belonging to the database partition on which the load operation is performed. For partitioned tables, all of the table spaces listed in SYSDATAPARTITIONS.TBSPACEID and SYSDATAPARTITIONS.LONG_TBSPACEID associated with a table and with a status of normal, attached or detached, (for example, SYSDATAPARTITIONS.STATUS of '', 'A' or 'D', respectively) are quiesced.

Authorization

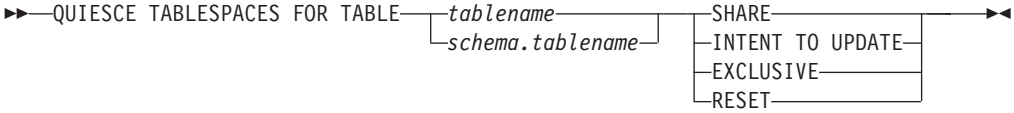
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM
- LOAD

Required connection

Database

Command syntax



Command parameters

TABLE

tablename
Specifies the unqualified table name. The table cannot be a system catalog table.

schema.tablename
Specifies the qualified table name. If *schema* is not provided, the CURRENT SCHEMA will be used. The table cannot be a system catalog table.

SHARE
Specifies that the quiesce is to be in share mode.

When a "quiesce share" request is made, the transaction requests intent share locks for the table spaces and a share lock for the table. When the transaction obtains the locks, the state of the table spaces is changed to

QUIESCED SHARE. The state is granted to the quiescer only if there is no conflicting state held by other users. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table, so that the state is persistent. The table cannot be changed while the table spaces for the table are in QUIESCED SHARE state. Other share mode requests to the table and table spaces are allowed. When the transaction commits or rolls back, the locks are released, but the table spaces for the table remain in QUIESCED SHARE state until the state is explicitly reset.

INTENT TO UPDATE
Specifies that the quiesce is to be in intent to update mode.

When a "quiesce intent to update" request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces is recorded in the table space table.

EXCLUSIVE
Specifies that the quiesce is to be in exclusive mode.

When a "quiesce exclusive" request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer) has exclusive access to the table and the table spaces.

RESET
Specifies that the state of the table spaces is to be reset to normal. A quiesce state cannot be reset if the connection that issued the quiesce request is still active.

Example

Quiesce the table spaces containing the staff table.

```
CALL SYSPROC.ADMIN_CMD( 'quiesce tablespaces for table staff share' )
```

Usage notes

This command is not supported for declared temporary tables.

A quiesce is a persistent lock. Its benefit is that it persists across transaction failures, connection failures, and even across system failures (such as power failure, or reboot).

A quiesce is owned by a connection. If the connection is lost, the quiesce remains, but it has no owner, and is called a *phantom quiesce*. For example, if a power outage caused a load operation to be interrupted during the delete phase, the table spaces for the loaded table would be left in delete pending, quiesce exclusive state. Upon database restart, this quiesce would be an unowned (or phantom) quiesce. The removal of a phantom quiesce requires a connection with the same user ID used when the quiesce mode was set.

To remove a phantom quiesce:

1. Connect to the database with the same user ID used when the quiesce mode was set.
2. Use the LIST TABLESPACES command to determine which table space is quiesced.
3. Re-quiesce the table space using the current quiesce state. For example:
   ```
   CALL SYSPROC.ADMIN_CMD('quiesce tablespaces for table mytable exclusive' )
   ```

Once completed, the new connection owns the quiesce, and the load operation can be restarted.

There is a limit of five quiescers on a table space at any given time.

A quiescer can upgrade the state of a table space from a less restrictive state to a more restrictive one (for example, S to U, or U to X). If a user requests a state lower than one that is already held, the original state is returned. States are not downgraded.

Command execution status is returned in the SQLCA resulting from the CALL statement.

REDISTRIBUTE DATABASE PARTITION GROUP command using the ADMIN_CMD procedure

Redistributes data across all partitions in a database partition group. This command affects all objects present in the database partition group and cannot be restricted to one object alone.

Scope

This command affects all database partitions in the database partition group.

Authorization

One of the following authorities is required:

- SYSADM
- SYSCTRL
- DBADM

In addition, one of the following groups of authorizations is also required:

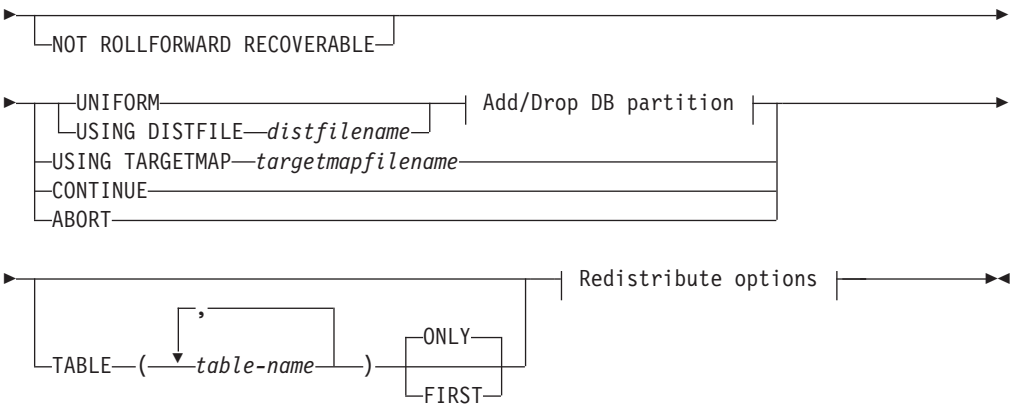
- DELETE, INSERT, and SELECT privileges on all tables in the database partition group being redistributed
- DATAACCESS authority

Required connection

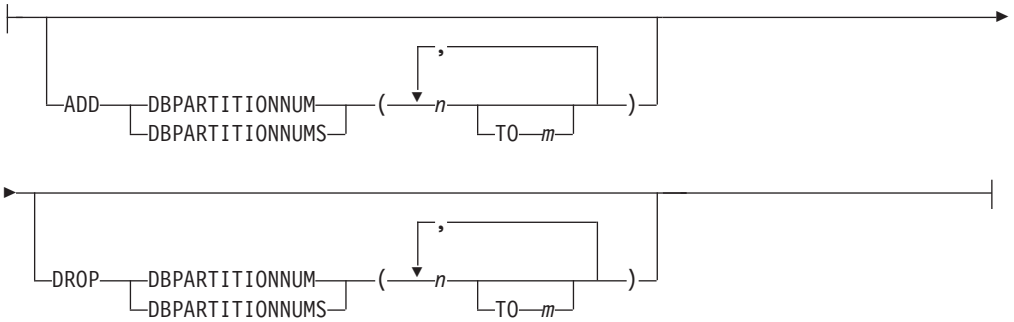
Connection to the catalog partition.

Command syntax

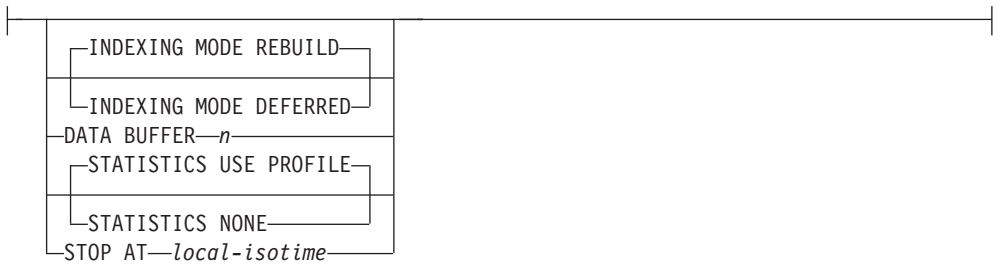
```
►►—REDISTRIBUTE DATABASE PARTITION GROUP—db-partition-group————————————►
```



Add/Drop DB partition:



Redistribute options:



Command parameters

DATABASE PARTITION GROUP *db-partition-group*
The name of the database partition group. This one-part name identifies a database partition group described in the SYSCAT.DBPARTITIONGROUPS catalog table. The database partition group cannot currently be undergoing redistribution.

Note: Tables in the IBMCATGROUP and the IBMTEMPGROUP database partition groups cannot be redistributed.

NOT ROLLFORWARD RECOVERABLE
When this option is used, the REDISTRIBUTE DATABASE PARTITION GROUP command is not roll forward recoverable.

- Data is moved in bulk instead of by internal insert and delete operations. This reduces the number of times that a table must be scanned and accessed, which results in better performance.
- Log records are no longer required for each of the insert and delete operations. This means that you no longer need to manage large amounts of active log space and log archiving space in your system when performing data redistribution. This is particularly beneficial if, in the past, large active log space and storage requirements forced you to break a single data redistribution operation into multiple smaller redistribution tasks, which might have resulted in even more time required to complete the end-to-end data redistribution operation.
- When using the REDISTRIBUTE DATABASE PARTITION GROUP command with the **NOT ROLLFORWARD RECOVERABLE** option, the redistribute operation uses the **INDEXING MODE DEFERRED** option for tables that contain XML columns. If a table does not contain an XML column, the redistribute operation uses the indexing mode specified when issuing the command.

When this option is *not* used, extensive logging of all row movement is performed such that the database can be recovered later in the event of any interruptions, errors, or other business need.

UNIFORM
Specifies that the data is uniformly distributed across hash partitions (that is, every hash partition is assumed to have the same number of rows), but the same number of hash partitions do not map to each database partition. After redistribution, all database partitions in the database partition group have approximately the same number of hash partitions.

USING DISTFILE *distfilename*
If the distribution of distribution key values is skewed, use this option to achieve a uniform redistribution of data across the database partitions of a database partition group.

Use the *distfilename* to indicate the current distribution of data across the 32 768 hash partitions.

Use row counts, byte volumes, or any other measure to indicate the amount of data represented by each hash partition. The utility reads the integer value associated with a partition as the weight of that partition. When a *distfilename* is specified, the utility generates a target distribution map that it uses to redistribute the data across the database partitions in the database partition group as uniformly as possible. After the redistribution, the weight of each database partition in the database partition group is approximately the same (the weight of a database partition is the sum of the weights of all hash partitions that map to that database partition).

For example, the input distribution file might contain entries as follows:

```
10223
1345
112000
0
100
...
```

In the example, hash partition 2 has a weight of 112000, and partition 3 (with a weight of 0) has no data mapping to it at all.

The *distfilename* should contain 32 768 positive integer values in character format. The sum of the values should be less than or equal to 4 294 967 295.

The complete path name for *distfilename* must be included and *distfilename* must exist on the server and be accessible from the connected partition.

USING TARGETMAP *targetmapfilename*
The file specified in *targetmapfilename* is used as the target distribution map. Data redistribution is done according to this file. The complete path name for *targetmapfilename* must be included and *targetmapfilename* must exist on the server and be accessible from the connected partition.

If a database partition, included in the target map, is not in the database partition group, an error is returned. Issue ALTER DATABASE PARTITION GROUP ADD DBPARTITIONNUM statement before running REDISTRIBUTE DATABASE PARTITION GROUP command.

If a database partition, excluded from the target map, *is* in the database partition group, that database partition will not be included in the partitioning. Such a database partition can be dropped using ALTER DATABASE PARTITION GROUP DROP DBPARTITIONNUM statement either before or after the REDISTRIBUTE DATABASE PARTITION GROUP command.

CONTINUE
Continues a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ABORT
Aborts a previously failed or stopped REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ADD

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be added into the database partition group. Any specified partition must not already be defined in the database partition group (SQLSTATE 42728). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with ADD DBPARTITIONNUM clause specified.

Note: When a database partition is added using this option, containers for table spaces are based on the containers of the corresponding table space on the lowest numbered existing partition in the database partition group. If this would result in a

naming conflict among containers, which could happen if the new partitions are on the same physical machine as existing containers, this option should not be used. Instead, the ALTER DATABASE PARTITION GROUP statement should be used with the WITHOUT TABLESPACES option prior to issuing the REDISTRIBUTE DATABASE PARTITION GROUP command. Table space containers can then be created manually specifying appropriate names.

DROP

DBPARTITIONNUM *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

DBPARTITIONNUMS *n*

TO *m*

n or *n* **TO** *m* specifies a list or lists of database partition numbers which are to be dropped from the database partition group. Any specified partition must already be defined in the database partition group (SQLSTATE 42729). This is equivalent to executing the ALTER DATABASE PARTITION GROUP statement with the DROP DBPARTITIONNUM clause specified.

TABLE *tablename*
Specifies a table order for redistribution processing.

ONLY If the table order is followed by the **ONLY** keyword (which is the default), then, only the specified tables will be redistributed. The remaining tables can be later processed by subsequent REDISTRIBUTE CONTINUE commands. This is the default.

FIRST If the table order is followed by the **FIRST** keyword, then, the specified tables will be redistributed with the given order and the remaining tables in the database partition group will be redistributed with random order.

INDEXING MODE
This parameter specifies how indexes are maintained during redistribution when the **NOT ROLLFORWARD RECOVERABLE** option is specified. Valid values are:

REBUILD
Indexes will be rebuilt from scratch. Indexes do not have to be valid to use this option. As a result of using this option, index pages will be clustered together on disk.

DEFERRED
Redistribute will not attempt to maintain any indexes. Indexes will be marked as needing a refresh. The first access to such indexes may force a rebuild, or indexes may be rebuilt when the database is restarted.

Note: For non-MDC tables, if there are invalid indexes on the tables, the REDISTRIBUTE DATABASE PARTITION GROUP command automatically rebuilds them if you do not specify **INDEXING MODE DEFERRED**. For an MDC table, even if you specify **INDEXING MODE DEFERRED**, a composite index that is invalid is rebuilt before table redistribution begins because the utility needs the composite index to process an MDC table.

DATA BUFFER *n*

Specifies the number of 4 KB pages to use as buffered space for transferring data within the utility. If the value specified is lower than the minimum supported value, the minimum value is used and no warning is returned. If a DATA BUFFER value is not specified, an intelligent default is calculated by the utility at runtime at the beginning of processing each table. Specifically, the default is to use 50% of the memory available in the utility heap at the time redistribution of the table begins and to take into account various table properties as well.

This memory is allocated directly from the utility heap, whose size can be modified through the **util_heap_sz** database configuration parameter. Beginning in version 9.5, the value of the DATA BUFFER option of the REDISTRIBUTE DATABASE PARTITION GROUP command can temporarily exceed **util_heap_sz** if more memory is available in the system.

STOP AT *local-isotime*

When this option is specified, before beginning data redistribution for each table, the *local-isotime* is compared with the current local timestamp. If the specified *local-isotime* is equal to or earlier than the current local timestamp, the utility stops with a warning message. Data redistribution processing of tables in progress at the stop time will complete without interruption. No new data redistribution processing of tables begins. The unprocessed tables can be redistributed using the **CONTINUE** option. This *local-isotime* value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds) expressed in local time.

STATISTICS

This option specifies that the utility should collect statistics for the tables that have a statistics profile. Specifying this option is more efficient than separately issuing the RUNSTATS command after the data redistribution is completed.

USE PROFILE

Statistics will be collected for the tables with a statistics profile. For tables without a statistics profile, nothing will be done. This is the default.

NONE

Statistics will not be collected for tables.

Examples: Redistribute steps

You may want to add or drop node from node group. Following is the step for adding new node to a node group and redistribute the data. Added database partition is not in the distribution map, but the containers for the table spaces in the database partition group have been created; the database partition is added to the distribution map when a redistribute database partition group operation has completed successfully.

1. Identify the nodegroups that will require redistribution. In this document, the node group that needs to be redistributed is “sampleNodegrp”.
2. Identify objects that should be disabled or removed before redistribute .
   a. Replicate MQTs: This type of MQT is not supported as part of the REDISTRIBUTE utility. They need to be dropped before running redistribute and recreated afterward.

      ```
      SELECT tabschema, tabname
        FROM syscat.tables
        WHERE partition_mode = 'R'
      ```

   b. Write-to-table event monitors: You should disable any automatically activated write-to-table event monitors that have a table that resides in the database partition group to be redistributed.

      ```
      SELECT distinct evmonname
        FROM syscat.eventtables E
        JOIN syscat.tables T on T.tabname = E.tabname
          AND T.tabschema = E.tabschema
        JOIN syscat.tablespaces S on S.tbspace = T.tbspace
          AND S.ngname = 'sampleNodegrp'
      ```

   c. Explain tables: It is recommended to create the explain tables in a single partition nodegroup. However, if they are defined in a nodegroup that requires redistribution, you may consider dropping them before the redistribute and redefining them once redistribute is complete, if the data generated to date does not need to be maintained.
   d. Table access mode and load state: Ensure that all tables in the node groups to be redistributed are in full access mode and have no load pending or load in progress state.

      ```
      SELECT DISTINCT TRIM(T.OWNER) || \'.\' || TRIM(T.TABNAME)
        AS NAME, T.ACCESS_MODE, A.LOAD_STATUS
       FROM SYSCAT.TABLES T, SYSCAT.DBPARTITIONGROUPS
        N, SYSIBMADM.ADMINTABINFO A
       WHERE T.PMAP_ID = N.PMAP_ID
        AND A.TABSCHEMA = T.OWNER
        AND A.TABNAME = T.TABNAME
        AND N.DBPGNAME = 'sampleNodegrp'
        AND (T.ACCESS_MODE <> 'F' OR A.LOAD_STATUS IS NOT NULL)
      ```

   e. Statistics profiles: Table statistics can be updated as part of the redistribution process if a statistics profile is defined for the table. Having the REDISTRIBUTE utility update a table's statistics reduces I/O as all the data is scanned for the redistribute and no additional scan of the data is needed for RUNSTATS.

      ```
      RUNSTATS on table schema.table
        USE PROFILE runstats_profile
        SET PROFILE ONLY
      ```

3. Review the database configuration. **util_heap_sz** is critical to the data movement processing between database partitions – allocate as much memory as possible to **util_heap_sz** for the duration of the redistribution. Sufficient **sortheap** is required, if index rebuild is done as part of the redistribution. Increase **util_heap_sz** and **sortheap** as necessary to improve redistribute performance.
4. Retrieve the database configuration settings to be used for the new database partitions. When adding database partitions, a default database configuration is used. As a result, it's important to update the database configuration on the new nodes before the REDISTRIBUTE command is issued to ensure that the configuration is balanced across the entire warehouse.

```
SELECT name,
  CASE WHEN deferred_value_flags = 'AUTOMATIC'
    THEN deferred_value_flags
    ELSE substr(deferred_value,1,20)
    END
  AS deferred_value
  FROM sysibmadm.dbcfg
  WHERE dbpartitionnum = existing-node
    AND deferred_value != ''
    AND name NOT IN ('hadr_local_host','hadr_local_svc','hadr_peer_window',
                 'hadr_remote_host','hadr_remote_inst','hadr_remote_svc',
                 'hadr_syncmode','hadr_timeout','backup_pending','codepage',
                 'codeset','collate_info','country','database_consistent',
                 'database_level','hadr_db_role','log_retain_status',
                 'loghead','logpath','multipage_alloc','numsegs','pagesize',
                 'release','restore_pending','restrict_access',
                 'rollfwd_pending','territory','user_exit_status',
                 'number_compat','varchar2_compat','database_memory')
```

5. Backup the database (or the table spaces in nodegroups that will be redistributed), before starting the redistribution process to ensure a recent recovery point.
6. Define the new data BCUs in DB2 by updating the `db2nodes.cfg` file and adding the new data BCU database partition specifications and define the new database partitions to DB2 using the ADD NODE WITHOUT TABLESPACES command.

   ```
   db2start nodenum x export DB2NODE=x
   db2 add node without tablespaces
   db2stop nodenum x
   ```

   Note: If it is not the first logical port on the data BCU, then execute a start and stop of the first logical port number before and after the above sequence of commands for subsequent logical ports.
7. Define system temporary table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name
     ADD container_information
     ON dbpartitionnums (x to y)
   ```
8. Add the new logical database partitions to the database partition groups that span the data BCUs.

   ```
   ALTER DATABASE PARTITION GROUP partition_group_name
     ADD dbpartitionnums (x to y)
     WITHOUT TABLESPACES
   ```
9. Define permanent data table space containers on the newly defined database partitions.

   ```
   ALTER TABLESPACE tablespace_name
     ADD container_information
     ON dbpartitionnums (x to y)
   ```
10. Apply the database configuration settings retrieved in step 4 to the new database partitions (or issue a single UPDATE DB CFG command against all database partitions using the new DB2 9.5 single view of configuration support).
11. Capture the definition of and then drop any replicated MQTs existing in the database partition groups to be redistributed.

    ```
    db2look -d dbname -e -z
      schema -t replicated_MQT_table_names
      -o repMQTs.clp
    ```
12. Disable any write-to-table event monitors that exist in the database partition groups to be redistributed.

```
SET EVENT MONITOR monitor_name STATE 0
```

13. Run the REDISTRIBUTE utility to redistribute uniformly across all database partitions. Following shows the simple redistribute command:

```
REDISTRIBUTE DATABASE PARTITION GROUP sampleNodegrp
  NOT ROLLFORWARD RECOVERABLE uniform;
```

User also should consider specifying a table list as input to the REDISTRIBUTE command to enforce the order that the tables will be processed. The REDISTRIBUTE utility will move the data (compressed and compacted). Optionally, indexes will be rebuilt and statistics updated if statistics profiles are defined. Therefore instead of previous command, the following script can be run:

```
REDISTRIBUTE DATABASE PARTITION GROUP sampleNodegrp
  NOT ROLLFORWARD RECOVERABLE uniform
  TABLE (tab1, tab2,...) FIRST;
```

Consequences of using the NOT ROLLFORWARD RECOVERABLE option

When the REDISTRIBUTE DATABASE PARTITION GROUP command is issued and the **NOT ROLLFORWARD RECOVERABLE** option is specified, a minimal logging strategy is used that minimizes the writing of log records for each moved row. This type of logging is important for the usability of the redistribute operation since an approach that fully logs all data movement could, for large systems, require an impractical amount of active and permanent log space and would generally have poorer performance characteristics. It is important, however, for users to be aware that as a result of this minimal logging model, the REDISTRIBUTE DATABASE PARTITION GROUP command is *not* rollforward recoverable. This means that any operation that results in the database rolling forward through a redistribute operation results in all tables touched by the redistribution operation being left in the UNAVAILABLE state. Such tables can only be dropped, which means there is no way to recover the data in these tables. This is why, for recoverable databases, the REDISTRIBUTE DATABASE PARTITION GROUP utility when issued with the NOT ROLLFORWARD RECOVERABLE option puts all table spaces it touches into the BACKUP PENDING state, forcing the user to backup all redistributed table spaces at the end of a successful redistribute operation. With a backup taken after the redistribution operation, the user should not have a need to rollforward through the redistribute operation itself.

There is one very important consequence of the redistribute utility's lack of rollforward recoverability of which the user should be aware: If the user chooses to allow updates to be made against tables in the database (even tables outside the database partition group being redistributed) while the redistribute operation is running, including the period at the end of redistribute where the table spaces touched by redistribute are being backed up by the user, such updates can be lost in the event of a serious failure, for example, a database container is destroyed. The reason that such updates can be lost is that the redistribute operation is not rollforward recoverable. If it is necessary to restore the database from a backup taken prior to the redistribution operation, then it will not be possible to rollforward through the logs in order to replay the updates that were made during the redistribution operation without also rolling forward through the redistribute which, as was described above, leaves the redistributed tables in the UNAVAILABLE state. Thus, the only thing that can be done in this situation is to restore the database from the backup taken prior to redistribute without rolling forward. Then the redistribute operation can be performed again. Unfortunately, all the updates that occurred during the original redistribute operation are lost.

The importance of this point cannot be overemphasized. In order to be certain that there will be no lost updates during a redistribution operation, one of the following must be true:

- The user avoids making updates during the operation of the REDISTRIBUTE DATABASE PARTITION GROUP command, including the period after the command finishes where the affected table spaces are being backed up.
- Updates that are applied during the redistribute operation come from a repeatable source, meaning that they can be applied again at any time. For example, if the source of updates is data that is stored in a file and the updates are applied during batch processing, then clearly even in the event of a failure requiring a database restore, the updates would not be lost since they could simply be applied again at any time.

With respect to allowing updates to the database during the redistribution operation, the user must decide whether such updates are appropriate or not for their scenario based on whether or not the updates can be repeated after a database restore, if necessary.

Note: Not every failure during operation of the REDISTRIBUTE DATABASE PARTITION GROUP command results in this problem. In fact, most do not. The REDISTRIBUTE DATABASE PARTITION GROUP command is fully restartable, meaning that if the utility fails in the middle of its work, it can be easily continued or aborted with the **CONTINUE** or **ABORT** options. The failures mentioned above are failures that require the user to restore from the backup taken prior to the redistribute operation.

Examples

Redistribute database partition group DBPG_1 by providing the current data distribution through a data distribution file, `distfile_for_dbpg_1`, and moving data onto two new database partitions, 6 and 7.

```
CALL SYSPROC.ADMIN_CMD('REDISTRIBUTE DATABASE PARTITION GROUP DBPG_1
   USING DISTFILE /home/user1/data/distfile_for_dbpg_1
   ADD DATABASE PARTITION (6 TO 7) ')
```

Usage notes

- Before starting a redistribute operation, ensure that the tables are in normal state and not in load pending state or reorg pending state. Table states can be checked by using the LOAD QUERY command.
- When the **NOT ROLLFORWARD RECOVERABLE** option is specified and the database is a recoverable database, the first time the utility accesses a table space, it is put into the BACKUP PENDING state. All the tables in that table space will become read-only until the table space is backed-up, which can only be done when all tables in the table space have finished being redistributed.
- When a redistribution operation is running, it produces an event log file containing general information about the redistribution operation and information such as the starting and ending time of each table processed. This event log file is written to the server:
  - The `homeinst/sqllib/redist` directory on Linux and UNIX operating systems, using the following format for subdirectories and file name: *database-name*`.`*database-partition-group-name*`.`*timestamp*`.log`.
  - The **DB2INSTPROF**`\`*instance*`\redist` directory on Windows operating systems (where **DB2INSTPROF** is the value of the **DB2INSTPROF** registry variable),

using the following format for subdirectories and file name: *database-name.database-partition-group-name.timestamp*.log.
  - The time stamp value is the time when the command was issued.
- This utility performs intermittent COMMITs during processing. This can cause type 2 connections to receive an SQL30090N error.
- All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.
- By default, the redistribute utility will update the statistics for those tables that have a statistics profile. For the tables without a statistics profile, it is recommended that you separately update the table and index statistics for these tables by calling the db2Runstats API or by issuing the RUNSTATS command after the redistribute operation has completed.
- Database partition groups containing replicated materialized query tables or tables defined with DATA CAPTURE CHANGES cannot be redistributed.
- Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables or created temporary tables in the database partition group.
- Options such as **INDEXING MODE** are ignored on tables, on which they do not apply, without warning. For example, **INDEXING MODE** will be ignored on tables without indexes.
- Command execution status is returned in the SQLCA resulting from the CALL statement.
- The file referenced in **USING DISTFILE** *distfilename* or **USING TARGETMAP** *targetmapfilename*, must refer to a file on the server.
- The REDISTRIBUTE DATABASE PARTITION GROUP command might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This command might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Compatibilities

Tables containing XML columns that use the DB2 Version 9.5 or earlier XML record format cannot be redistributed. Use the ADMIN_MOVE_TABLE stored procedure to migrate the table to the new format.

For compatibility with versions earlier than Version 8:

- The keyword **NODEGROUP** can be substituted for **DATABASE PARTITION GROUP**.

REORG INDEXES/TABLE command using the ADMIN_CMD procedure

Reorganizes an index or a table.

You can reorganize all indexes defined on a table by rebuilding the index data into unfragmented, physically contiguous pages. On a data partitioned table, you can reorganize a specific nonpartitioned index on a partitioned table, or you can reorganize all the partitioned indexes on a specific data partition.

If you specify the **CLEANUP ONLY** option of the index clause, cleanup is performed without rebuilding the indexes. This command cannot be used against indexes on declared temporary tables or created temporary tables (SQLSTATE 42995).

The table option reorganizes a table by reconstructing the rows to eliminate fragmented data, and by compacting information. On a partitioned table, you can reorganize a single partition.

Scope

This command affects all database partitions in the database partition group.

Authorization

One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM
- SQLADM
- CONTROL privilege on the table.

Required connection

Database

Command syntax

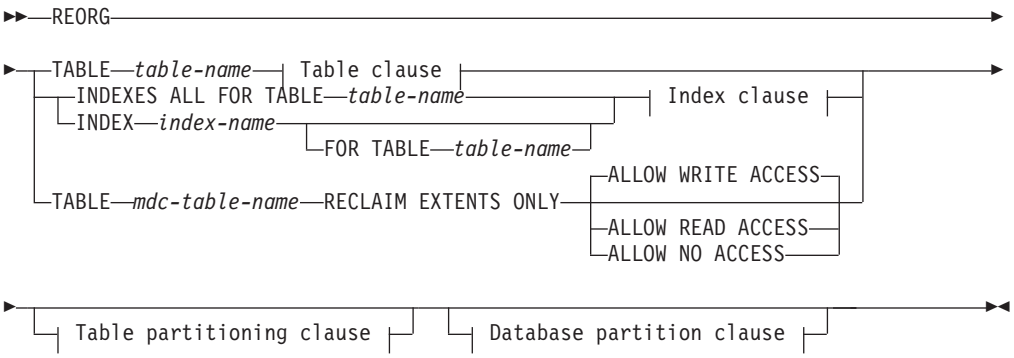
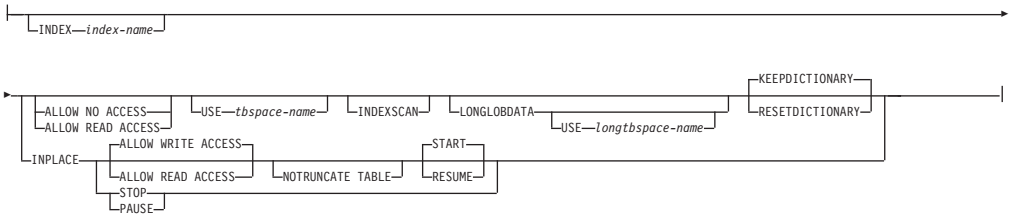


Table clause:



Index clause:

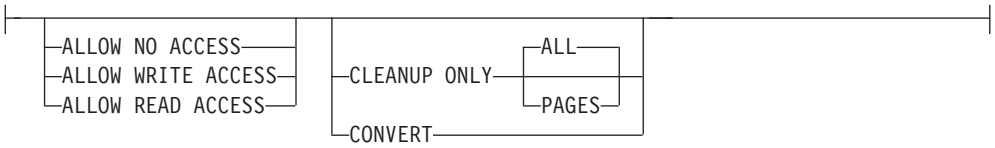
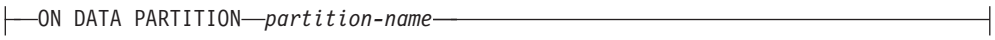
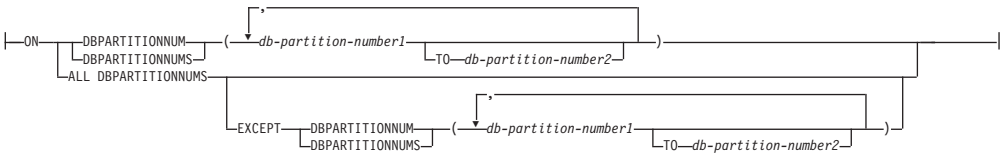


Table partitioning clause:



Database partition clause:



Command parameters

INDEXES ALL FOR TABLE *table-name*
Specifies the table whose indexes are to be reorganized. The table can be in a local or a remote database.

INDEX *index-name*
Specifies an individual index to be reorganized on a data partitioned table. Reorganization of individual indexes are *only* supported for nonpartitioned indexes on a partitioned table. This parameter is not supported for block indexes.

FOR TABLE *table-name*
Specifies the name of the table on which the nonpartitioned index *index-name* is created. This parameter is optional, given that index names are unique across the database.

ALLOW NO ACCESS
For REORG INDEXES, specifies that no other users can access the table while the indexes are being reorganized. If the **ON DATA PARTITION** clause is specified for a partitioned table, only the specified partition is restricted to the access mode level.

For REORG INDEX, specifies that no other users can access the table while the nonpartitioned index is being reorganized.

ALLOW READ ACCESS
For REORG INDEXES, specifies that other users can have read-only access to the table while the indexes are being reorganized. **ALLOW READ ACCESS** mode is not supported for REORG INDEXES of a partitioned table unless the **CLEANUP ONLY** option or **ON DATA PARTITION** clause is specified. If the **ON DATA PARTITION** clause is specified for a partitioned table, only the specified partition is restricted to the access mode level.

For REORG INDEX, specifies that can have read-only access to the table while the nonpartitioned index is being reorganized.

ALLOW WRITE ACCESS

For REORG INDEXES, specifies that other users can read from and write to the table while the indexes are being reorganized. **ALLOW WRITE ACCESS** mode is not supported for a partitioned table unless the **CLEANUP ONLY** option or **ON DATA PARTITION** clause is specified. If the **ON DATA PARTITION** clause is specified for a partitioned table, only the specified partition is restricted to the access mode level.

For REORG INDEX, specifies that can read from and write to the table while the nonpartitioned index is being reorganized.

ALLOW WRITE ACCESS mode is not supported for multidimensional clustering (MDC) tables or extended indexes unless the **CLEANUP ONLY** option is specified.

The following items apply for a data partitioned table when the **ON DATA PARTITION** clause is specified with the REORG INDEXES ALL command:

- Only the specified data partition is restricted to the access mode level. Users are allowed to read from and write to the other partitions of the table while the partitioned indexes of a specified partition are being reorganized.

  The following table lists the access modes supported and the concurrent access allowed on other partitions of the table when the **ON DATA PARTITION** clause is specified:

Table 61. Access modes supported and concurrent access allowed when the ON DATA PARTITION clause is specified with REORG INDEXES ALL

Access mode	Concurrent access allowed on the specified partition	Concurrent access allowed on other partitions
ALLOW NO ACCESS	No access	Read and write access
ALLOW READ ACCESS	Read on the partition up until index is updated	Read and write access
ALLOW WRITE ACCESS	Read and write access on the partition up until index is updated	Read and write access

- Only the partitioned indexes for the specified partition are reorganized. The nonpartitioned indexes on the partitioned table are not reorganized.

  If there are any nonpartitioned indexes on the table marked "invalid" or "for rebuild", all indexes marked "invalid" or "for rebuild" are rebuilt before reorganization. Otherwise, only the partitioned indexes on the specified partition are reorganized or rebuilt if the index object is marked "invalid" or "for rebuild".
- Only partitioned indexes for the specified partition are cleaned when the **CLEANUP ONLY** option is also specified.

The following table lists the supported access modes for index reorganization of partitioned and nonpartitioned tables:

Table 62. Supported access modes for index reorganization on partitioned and nonpartitioned table

Command	Table type	Table partitioning clause	Additional parameters specified for index clause	Supported access mode
REORG INDEXES	Nonpartitioned table	Not applicable	Any	**ALLOW NO ACCESS**, **ALLOW READ ACCESS**[1], **ALLOW WRITE ACCESS**
REORG INDEX	Partitioned table	Not applicable	Any	**ALLOW READ ACCESS**[1]
REORG INDEXES	Partitioned table	None	None specified	**ALLOW NO ACCESS** [1]
REORG INDEXES	Partitioned table	**ON DATA PARTITION**	None specified	**ALLOW NO ACCESS**, **ALLOW READ ACCESS**[1], **ALLOW WRITE ACCESS**
REORG INDEXES	Partitioned table	With or without the **ON DATA PARTITION** clause	**CLEANUP ONLY** specified	**ALLOW NO ACCESS**, **ALLOW READ ACCESS**[1], **ALLOW WRITE ACCESS**

Note:

1. Default mode when an access clause is not specified.

CLEANUP ONLY

When **CLEANUP ONLY** is requested, a cleanup rather than a full reorganization will be done. The indexes will not be rebuilt and any pages freed up will be available for reuse by indexes defined on this table only.

The **CLEANUP ONLY PAGES** option will search for and free committed pseudo empty pages. A committed pseudo empty page is one where all the keys on the page are marked as deleted and all these deletions are known to be committed. The number of pseudo empty pages in an indexes can be determined by running RUNSTATS and looking at the NUM EMPTY LEAFS column in SYSCAT.INDEXES. The **PAGES** option will clean the NUM EMPTY LEAFS if they are determined to be committed.

The **CLEANUP ONLY ALL** option will free committed pseudo empty pages, as well as remove committed pseudo deleted keys from pages that are not pseudo empty. This option will also try to merge adjacent leaf pages if doing so will result in a merged leaf page that has at least PCTFREE free space on the merged leaf page, where PCTFREE is the percent free space defined for the index at index creation time. The default PCTFREE is ten percent. If two pages can be merged, one of the pages will be freed. The number of pseudo deleted keys in an index , excluding those on pseudo empty pages, can be determined by running RUNSTATS and then selecting the NUMRIDS DELETED from SYSCAT.INDEXES. The **ALL** option will clean the NUMRIDS DELETED and the NUM EMPTY LEAFS if they are determined to be committed.

ALL Specifies that indexes should be cleaned up by removing committed pseudo deleted keys and committed pseudo empty pages.

PAGES

Specifies that committed pseudo empty pages should be removed from the index tree. This will not clean up pseudo deleted keys on

pages that are not pseudo empty. Since it is only checking the pseudo empty leaf pages, it is considerably faster than using the **ALL** option in most cases.

CONVERT
Converts type-1 indexes to type-2 index. If the index is already type 2, this option has no effect.

In Version 9.7, type-1 indexes are discontinued and all indexes that are created are type-2 indexes. As a result, the **CONVERT** option is deprecated.

All indexes created prior to Version 8 are type-1 indexes. Prior to Version 9.7, all indexes created by Version 8 and later are type-2 indexes, except when you create an index on a table that already has a type-1 index. In this case, the new index was also of type 1. This is no longer the case in Version 9.7 because all indexes created are type 2.

Use the **ALLOW READ ACCESS** or **ALLOW WRITE ACCESS** option to allow other transactions either read-only or read-write access to the table while the indexes are being reorganized. While **ALLOW READ ACCESS** and **ALLOW WRITE ACCESS** allow access to the table, during the period in which the reorganized copies of the indexes are made available, no access to the table is allowed.

TABLE *mdc-table-name* **RECLAIM EXTENTS ONLY**
Specifies the multidimensional clustering (MDC) table to reorganize to reclaim extents that are not being used. The name or alias in the form: *schema.table-name* can be used. The *schema* is the user name under which the table was created. If you omit the schema name, the default schema is assumed.

For REORG TABLE RECLAIM EXTENTS ONLY when the **ON DATA PARTITION** clause is specified, the access clause only applies to the named partition. Users can read from and write to the rest of the table while the extents on the specified partition are being reclaimed. This situation also applies to the default access levels.

ALLOW NO ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that no other users can access the table while the extents are being reclaimed.

ALLOW READ ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that other users can have read-only access to the table while the extents are being reclaimed.

ALLOW WRITE ACCESS
For REORG TABLE RECLAIM EXTENTS ONLY, specifies that other users can read from and write to the table while the extents are being reclaimed.

TABLE *table-name*
Specifies the table to reorganize. The table can be in a local or a remote database. The name or alias in the form: *schema.table-name* can be used. The *schema* is the user name under which the table was created. If you omit the schema name, the default schema is assumed.

For typed tables, the specified table name must be the name of the hierarchy's root table.

You cannot specify an index for the reorganization of a multidimensional clustering (MDC) table. In place reorganization of tables cannot be used for MDC tables.

When the **ON DATA PARTITION** clause is specified for a table reorganization of a data partitioned table, only the specified data partition is reorganized:

- If there are no nonpartitioned indexes (except system-generated XML path indexes) defined on the table, the access mode applies only to the specified partition, users are allowed to read from and write to the other partitions of the table.
- If there are nonpartitioned indexes defined on the table (excluding system-generated XML path indexes), the **ALLOW NO ACCESS** mode is the default and only supported access mode. In this case, the table is placed in **ALLOW NO ACCESS** mode. If **ALLOW READ ACCESS** is specified, SQL1548N is returned (SQLSTATE 5U047).

Table 63. Supported access mode for table reorganization on nonpartitioned and partitioned table

Command	Table type	Table partitioning clause	Supported access mode
REORG TABLE	Nonpartitioned table	Not applicable	**ALLOW NO ACCESS**, **ALLOW READ ACCESS**[1]
REORG TABLE	Partitioned table	Not specified	**ALLOW NO ACCESS**[1]
REORG TABLE (There are no indexes or only partitioned indexes defined on the table.)	Partitioned table	**ON DATA PARTITION**	**ALLOW NO ACCESS**, **ALLOW READ ACCESS**[1]
REORG TABLE (there are nonpartitioned indexes defined on the table, excluding system-generated XML path indexes.)	Partitioned table	**ON DATA PARTITION**	**ALLOW NO ACCESS**[1]

Note:

1. Default mode when an access clause is not specified.

For a data partitioned table, a table reorganization rebuilds the nonpartitioned indexes and partitioned indexes on the table after reorganizing the table. If the **ON DATA PARTITION** clause is used to reorganize a specific data partition of a data partitioned table, a table reorganization rebuilds the nonpartitioned indexes and partitioned indexes only for the specified partition.

INDEX *index-name*
Specifies the index to use when reorganizing the table. If you do not specify the fully qualified name in the form: *schema.index-name*, the default schema is assumed. The *schema* is the user name under which the index was created. The database manager uses the index to physically reorder the records in the table it is reorganizing.

For an in place table reorganization, if a clustering index is defined on the table and an index is specified, it must be clustering index. If the in place option is not specified, any index specified will be used. If you do not specify the name of an index, the records are reorganized without regard to order. If the table has a clustering

index defined, however, and no index is specified, then the clustering index is used to cluster the table. You cannot specify an index if you are reorganizing an MDC table.

If a table reorganization uses both the **INDEX** and **ON DATA PARTITION** clauses, only the specified partition is reorganized using the index *index-name*.

ALLOW NO ACCESS
Specifies that no other users can access the table while the table is being reorganized.

The **ALLOW NO ACCESS** mode is the default and only supported access mode when reorganizing a partitioned table without the **ON DATA PARTITION** clause.

If the **ON DATA PARTITION** clause is specified for a data partitioned table, only the specified data partition is reorganized:

- If there are no nonpartitioned indexes defined on the table (except system-generated XML path indexes), only the specified partition is restricted to the **ALLOW NO ACCESS** mode. Users are allowed to read from and write to the other partitions of the table.
- If there are nonpartitioned indexes defined on the table (except system-generated XML path indexes), the **ALLOW NO ACCESS** mode is the default and only supported access mode. In this case, the table is placed in **ALLOW NO ACCESS** mode.

ALLOW READ ACCESS
Allow only read access to the table during reorganization.

The **ALLOW READ ACCESS** mode is the default mode for a nonpartitioned table.

If the **ON DATA PARTITION** clause is specified for a data partitioned table, only the specified data partition is reorganized:

- If there are no nonpartitioned indexes defined on the table (except system-generated XML path indexes), the **ALLOW READ ACCESS** mode is the default mode and only the specified partition is restricted to the access mode level. Users are allowed to read from and write to the other partitions of the table.
- If there are nonpartitioned indexes defined on the table (except system-generated XML path indexes), the **ALLOW READ ACCESS** mode is not supported. If **ALLOW READ ACCESS** is specified in this case, SQL1548N is returned (SQLSTATE 5U047)

INPLACE
Reorganizes the table while permitting user access.

In place table reorganization is allowed only on nonpartitioned and non-MDC tables with type-2 indexes, but without extended indexes and with no indexes defined over XML columns in the table. In place table reorganization can only be performed on tables that are at least three pages in size.

In place table reorganization takes place asynchronously, and might not be effective immediately.

ALLOW READ ACCESS
Allow only read access to the table during reorganization.

ALLOW WRITE ACCESS
Allow write access to the table during reorganization. This is the default behavior.

NOTRUNCATE TABLE
Do not truncate the table after in place reorganization. During truncation, the table is S-locked.

START
Start the in place REORG processing. Because this is the default, this keyword is optional.

STOP Stop the in place REORG processing at its current point.

PAUSE
Suspend or pause in place REORG for the time being.

RESUME
Continue or resume a previously paused in place table reorganization. When an online reorganization is resumed and you want the same options as when the reorganization was paused, you must specify those options again while resuming.

USE *tbspace-name*
Specifies the name of a system temporary table space in which to store a temporary copy of the table being reorganized. If you do not provide a table space name, the database manager stores a working copy of the table in the table spaces that contain the table being reorganized.

For an 8 KB, 16 KB, or 32 KB table object, if the page size of the system temporary table space that you specify does not match the page size of the table spaces in which the table data resides, the DB2 database product will try to find a temporary table space of the correct size of the LONG/LOB objects. Such a table space must exist for the reorganization to succeed.

For partitioned tables, the temporary table space is used as temporary storage for the reorganization of data partitions in the table. Reorganization of the entire partitioned table reorganizes a single data partition at a time. The temporary table space must be able to hold the largest data partition in the table, and not the entire table. When the **ON DATA PARTITION** clause is specified, the temporary table space must be able to hold the specified partition.

If you do not supply a table space name for a partitioned table, the table space where each data partition is located is used for temporary storage of that data partition. There must be enough free space in each data partition's table space to hold a copy of the data partition.

INDEXSCAN
For a clustering REORG an index scan will be used to re-order table records. Reorganize table rows by accessing the table through an index. The default method is to scan the table and sort the result to reorganize the table, using temporary table spaces as

necessary. Even though the index keys are in sort order, scanning and sorting is typically faster than fetching rows by first reading the row identifier from an index.

LONGLOBDATA
Long field and LOB data are to be reorganized.

This is not required even if the table contains long or LOB columns. The default is to avoid reorganizing these objects because it is time consuming and does not improve clustering. However, running a reorganization with the **LONGLOBDATA** option on tables with XML columns will reclaim unused space and thereby reduce the size of the XML storage object.

This parameter is required when converting existing LOB data into inlined LOB data.

USE *longtbspace-name*
This is an optional parameter, which can be used to specify the name of a temporary table space to be used for rebuilding long data. If no temporary table space is specified for either the table object or for the long objects, the objects will be constructed in the table space they currently reside. If a temporary table space is specified for the table but this parameter is not specified, then the table space used for base reorg data will be used, unless the page sizes differ. In this situation, the DB2 database system will attempt to choose a temporary container of the appropriate page size to create the long objects in.

If **USE** *longtbspace-name* is specified, **USE** *tbspace-name* must also be specified. If it is not, the *longtbspace-name* argument is ignored.

KEEPDICTIONARY
If the `COMPRESS` attribute for the table is `YES` and the table has a compression dictionary then no new dictionary is built. All the rows processed during reorganization are subject to compression using the existing dictionary. If the `COMPRESS` attribute is `YES` and a compression dictionary doesn't exist for the table, a dictionary will only be created (and the table compressed) in this scenario if the table is of a certain size (approximately 1 to 2 MB) and sufficient data exists within this table. If, instead, you explicitly state REORG RESETDICTIONARY, then a dictionary is built as long as there is at least 1 row in the table. If the `COMPRESS` attribute for the table is `NO` and the table has a compression dictionary, then reorg processing will preserve the dictionary and all the rows in the newly reorganized table will be in noncompressed format. It is not possible to compress some data such as LOB data not stored in the base table row.

When the **LONGLOBDATA** option is not specified, only the table row data is reorganized. The following table describes the behavior of **KEEPDICTIONARY** syntax in REORG command when the **LONGLOBDATA** option is not specified.

Table 64. REORG KEEPDICTIONARY

Compress	Dictionary Exists	Result; outcome
Y	Y	Preserve dictionary; rows compressed.
Y	N	Build dictionary; rows compressed

Table 64. REORG KEEPDICTIONARY (continued)

Compress	Dictionary Exists	Result; outcome
N	Y	Preserve dictionary; all rows uncompressed
N	N	No effect; all rows uncompressed

The following table describes the behavior of **KEEPDICTIONARY** syntax in REORG command when the **LONGLOBDATA** option is specified.

Table 65. REORG KEEPDICTIONARY when LONGLOBDATA option is specified.

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Compression dictionary	Data compression
Y	Y	Y	Preserve dictionaries.	Existing data is compressed. New data will be compressed.
Y	Y	N	Preserve table row dictionary and create an XML storage object dictionary.	Existing data is compressed. New data will be compressed.
Y	N	Y	Create table row dictionary and preserve the XML dictionary.	Existing data is compressed. New data will be compressed.
Y	N	N	Create table row and XML dictionaries.	Existing data is compressed. New data will be compressed.
N	Y	Y	Preserve table row and XML dictionaries.	Table data is uncompressed. New data will be not be compressed.
N	Y	N	Preserve table row dictionary.	Table data is uncompressed. New data will be not be compressed.
N	N	Y	Preserve XML dictionary.	Table data is uncompressed. New data will be not be compressed.
N	N	N	No effect.	Table data is uncompressed. New data will be not be compressed.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 V9.7 or later, or if the table is migrated using the ONLINE_TABLE_MOVE stored procedure.

For any reinitialization or truncation of a table (such as for a replace operation), if the compress attribute for the table is NO, the dictionary is discarded if one exists. Conversely, if a dictionary

exists and the compress attribute for the table is YES then a truncation will save the dictionary and not discard it. The dictionary is logged in its entirety for recovery purposes and for future support with data capture changes (that is, replication).

RESETDICTIONARY
If the COMPRESS attribute for the table is YES then a new row compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary. If the COMPRESS attribute for the table is NO and the table does have an existing compression dictionary then reorg processing will remove the dictionary and all rows in the newly reorganized table will be in noncompressed format. It is not possible to compress some data such as LOB data not stored in the base table row.

If the **LONGLOBDATA** option is not specified, only the table row data is reorganized. The following table describes the behavior of **RESETDICTIONARY** syntax in REORG command when the **LONGLOBDATA** option is not specified.

Table 66. REORG RESETDICTIONARY

Compress	Dictionary Exists	Result; outcome
Y	Y	Build new dictionary*; rows compressed. If DATA CAPTURE CHANGES option is specified on the CREATE TABLE or ALTER TABLE statements, the current dictionary is kept (referred to as the *historical compression dictionary*).
Y	N	Build new dictionary; rows compressed
N	Y	Remove dictionary; all rows uncompressed. If the DATA CAPTURE NONE option is specified on the CREATE TABLE or ALTER TABLE statements, the *historical compression dictionary* is also removed for the specified table.
N	N	No effect; all rows uncompressed

* - If a dictionary exists and the compression attribute is enabled but there currently isn't any data in the table, the **RESETDICTIONARY** operation will keep the existing dictionary. Rows which are smaller in size than the internal minimum record length and rows which do not demonstrate a savings in record length when an attempt is made to compress them are considered "insufficient" in this case.

The following table describes the behavior of **RESETDICTIONARY** syntax in REORG command when the **LONGLOBDATA** option is specified.

Table 67. REORG RESETDICTIONARY when LONGLOBDATA option is specified.

Compress	Table row data dictionary exists	XML storage object dictionary exists[1]	Data dictionary	Data compression
Y	Y	Y	Build dictionaries[2] [3].	Existing data is compressed. New data will be compressed.
Y	Y	N	Build new table row dictionary and create a new XML dictionary[3].	Existing data is compressed. New data will be compressed.
Y	N	Y	Create table row data dictionary and build a new XML dictionary.	Existing data is compressed. New data will be compressed.
Y	N	N	Create dictionaries.	Existing data is compressed. New data will be compressed.
N	Y	Y	Remove dictionaries. Existing and new data is not compressed.	Existing table data is uncompressed. New data will be not be compressed.
N	Y	N	Remove table row dictionary. All data is uncompressed.	Existing table data is uncompressed. New data will be not be compressed.
N	N	Y	Remove XML storage object dictionary.	Existing table data is uncompressed. New data will be not be compressed.
N	N	N	No effect.	Existing table data is uncompressed. New data will be not be compressed.

Note:

1. A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 V9.7 or later, or if the table is migrated using an online table move.
2. If a dictionary exists and the compression attribute is enabled but there currently isn't any data in the table, the **RESETDICTIONARY** operation will keep the existing dictionary. Rows which are smaller in size than the internal minimum record length and rows which do not demonstrate a savings in record length when an attempt is made to compress them are considered insufficient in this case.
3. If DATA CAPTURE CHANGES option is specified on the CREATE TABLE or ALTER TABLE statements, the current data dictionary is kept (referred to as the *historical compression dictionary*).

ON DATA PARTITION *partition-name*

For data partitioned tables, specifies the data partition for the reorganization.

For DB2 V9.7 Fix Pack 1 and later releases, the clause can be used with the REORG INDEXES ALL command to reorganize the partitioned indexes on a specific partition and the REORG TABLE command to reorganize data of a specific partition.

When using the clause with a REORG TABLE or REORG INDEXES ALL command on a partitioned table, the reorganization fails and returns SQL2222N with reason code 1 if the partition *partition-name* does not exist for the specified table. The reorganization fails and returns SQL2222N with reason code 3 if the partition *partition-name* is in the attached or detached state.

If the REORG INDEX command is issued with the **ON DATA PARTITION** clause, the reorganization fails and returns SQL2222N with reason code 2.

The REORG TABLE command fails and returns SQL1549N (SQLSTATE 5U047) if the partitioned table is in the reorg pending state and there are nonpartitioned indexes defined on the table.

ALL DBPARTITIONNUMS
Specifies that operation is to be done on all database partitions specified in the `db2nodes.cfg` file. This is the default if a node clause is not specified.

EXCEPT
Specifies that operation is to be done on all database partitions specified in the `db2nodes.cfg` file, except those specified in the node list.

ON DBPARTITIONNUM | ON DBPARTITIONNUMS
Perform operation on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

db-partition-number2
Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

Example

Reorganize the tables in a database partition group consisting of database partitions 1, 3 and 4.

```
CALL SYSPROC.ADMIN_CMD ('REORG TABLE employee
   INDEX empid ON DBPARTITIONNUM (1,3,4)')
```

Usage notes

Restrictions:

- Command execution status is returned in the SQLCA resulting from the CALL statement.
- The REORG utility issue a COMMIT statement at the beginning of the operation which, in the case of Type 2 connections, causes the procedure to return SQL30090N with reason code 2.
- The REORG utility does not support the use of nicknames.
- The REORG TABLE command is not supported for declared temporary tables or created temporary tables.
- The REORG TABLE command cannot be used on views.

- Reorganization of a table is not compatible with range-clustered tables, because the range area of the table always remains clustered.
- REORG TABLE cannot be used on a partitioned table in a DMS table space while an online backup of ANY table space in which the table resides, including LOBs and indexes, is being performed.
- REORG TABLE cannot use an index that is based on an index extension.
- If a table is in reorg pending state, an inplace reorg is not allowed on the table.
- Concurrent table reorganization sharing the same temporary DMS table space is not supported.
- For data partitioned tables:
  - The table must have an ACCESS_MODE in SYSCAT.TABLES of Full Access.
  - Reorganization skips data partitions that are in a restricted state due to an attach or detach operation. If the **ON DATA PARTITION** clause is specified, that partition must be fully accessible.
  - If an error occurs during table reorganization, some indexes or index partitions might be left invalid. The nonpartitioned indexes of the table will be marked invalid if the reorganization has reached or passed the replace phase for the first data partition. The index partitions for any data partition that has already reached or passed the replace phase will be marked invalid. Indexes will be rebuilt on the next access to the table or data partition.
  - If an error occurs during index reorganization when the **ALLOW NO ACCESS** mode is used, some indexes on the table might be left invalid. For nonpartitioned RID indexes on the table, only the index that is being reorganized at the time of the failure will be left invalid. For MDC tables with nonpartitioned block indexes, one or more of the block indexes might be left invalid if an error occurs. For partitioned indexes, only the index object on the data partition being reorganized will be left invalid. Any indexes marked invalid will be rebuilt on the next access to the table or data partition.
  - When a data partitioned table with only partitioned indexes defined on the table is in the reorg pending state, issuing a REORG TABLE command with the **ON DATA PARTITION** clause brings only the specified data partition out of the reorg pending state. To bring the remaining partitions of the table out of the reorg pending state, either issue REORG TABLE command on the entire table (without the **ON DATA PARTITION** clause), or issue a REORG TABLE command with the **ON DATA PARTITION** clause for each of the remaining partitions.

Information about the current progress of table reorganization is written to the history file for database activity. The history file contains a record for each reorganization event. To view this file, execute the LIST HISTORY command for the database that contains the table you are reorganizing.

You can also use table snapshots to monitor the progress of table reorganization. Table reorganization monitoring data is recorded regardless of the Database Monitor Table Switch setting.

If an error occurs, an SQLCA dump is written to the history file. For an inplace table reorganization, the status is recorded as PAUSED.

When an indexed table has been modified many times, the data in the indexes might become fragmented. If the table is clustered with respect to an index, the table and index can get out of cluster order. Both of these factors can adversely affect the performance of scans using the index, and can impact the effectiveness of

index page prefetching. REORG INDEX or REORG INDEXES can be used to reorganize one or all of the indexes on a table. Index reorganization will remove any fragmentation and restore physical clustering to the leaf pages. Use the REORGCHK command to help determine if an index needs reorganizing. Be sure to complete all database operations and release all locks before invoking index reorganization. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

A classic table reorganization (offline reorganization) rebuilds the indexes during the last phase of the reorganization. When more than one temporary table space exists, it is possible that a temporary table space in addition to the one specified on the REORG TABLE command may be utilized for additional sorts that can accompanying table reorg processing. However, the inplace table reorganization (online reorganization) does not rebuild the indexes. It is recommended that you issue a REORG INDEXES command after the completion of an inplace table reorganization. An inplace table reorganization is asynchronous, therefore care must be taken to ensure that the inplace table reorganization is complete before issuing the REORG INDEXES command. Issuing the REORG INDEXES command before the inplace table reorganization is complete, might cause the reorganization to fail (SQLCODE -2219).

Tables that have been modified so many times that data is fragmented and access performance is noticeably slow are candidates for the REORG TABLE command. You should also invoke this utility after altering the inline length of a structured type column in order to benefit from the altered inline length. Use the REORGCHK command to determine whether a table needs reorganizing. Be sure to complete all database operations and release all locks before invoking REORG TABLE. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK. After reorganizing a table, use RUNSTATS to update the table statistics, and REBIND to rebind the packages that use this table. The reorganize utility will implicitly close all the cursors.

With DB2 V9.7 Fix Pack 1 and later, REORG TABLE commands and REORG INDEXES ALL commands can be issued on a data partitioned table to concurrently reorganize different data partitions or partitioned indexes on a partition. When concurrently reorganizing data partitions or the partitioned indexes on a partition, users can access the unaffected partitions but cannot access the affected partitions. All the following criteria must be met to issue REORG commands that operate concurrently on the same table:

- Each REORG command must specify a different partition with the **ON DATA PARTITION** clause.
- Each REORG command must use the **ALLOW NO ACCESS** mode restrict access to the data partitions.
- The partitioned table must have only partitioned indexes if issuing REORG TABLE commands. No nonpartitioned indexes (except system-generated XML path indexes) can be defined on the table.

For a partitioned table T1 with no nonpartitioned indexes (except system-generated XML path indexes) and with partitions P1, P2, P3, and P4, the following REORG commands can run concurrently:

```
REORG INDEXES ALL ALLOW NO ACCESS ON DATA PARTITION P1
REORG TABLE ALLOW NO ACCESS ON DATA PARTITION P2
REORG INDEXES ALL ALLOW NO ACCESS ON DATA PARTITION P3
```

Operations such as the following are not supported when using concurrent REORG commands:

- Using a REORG command without the **ON DATA PARTITION** clause on the table.
- Using an ALTER TABLE statement on the table to add, attach, or detach a data partition.
- Loading data into the table.
- Performing an online backup that includes the table.

If the table contains mixed row format because the table value compression has been activated or deactivated, an offline table reorganization can convert all the existing rows into the target row format.

If the table is distributed across several database partitions, and the table or index reorganization fails on any of the affected database partitions, only the failing database partitions will have the table or index reorganization rolled back.

If the reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.

If the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries. If the name of an index is *not* specified, and if a clustering index exists, the data will be ordered according to the clustering index.

The PCTFREE value of a table determines the amount of free space designated per page. If the value has not been set, the utility will fill up as much space as possible on each page.

To complete a table space roll-forward recovery following a table reorganization, both regular and large table spaces must be enabled for roll-forward recovery.

If the table contains LOB columns that do not use the **COMPACT** option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS or DMS).

Indexes over XML data may be recreated by the REORG INDEXES/TABLE command. For details, see "Recreation of indexes over XML data".

RESET ALERT CONFIGURATION command using the ADMIN_CMD procedure

Resets the health indicator settings for specific objects to the current defaults for that object type or resets the current default health indicator settings for an object type to the install defaults.

Important: This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Authorization

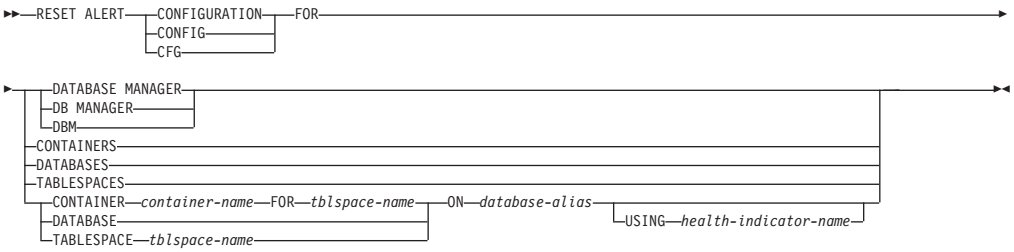
One of the following:

- SYSADM
- SYSMAINT
- SYSCTRL

Required connection

Database

Command syntax



Command parameters

DATABASE MANAGER | DB MANAGER | DBM
Resets alert settings for the database manager.

CONTAINERS
Resets default alert settings for all table space containers managed by the database manager to the install default. These are the settings that apply to all table space containers that do not have custom settings. Custom settings are defined using the **CONTAINER** *container-name* **FOR** *tblspace-name* **ON** *database-alias* clause.

DATABASES
Resets alert settings for all databases managed by the database manager. These are the settings that apply to all databases that do not have custom settings. Custom settings are defined using the **DATABASE ON** *database-alias* clause.

TABLESPACES
Resets default alert settings for all table spaces managed by the database manager to the install default. These are the settings that apply to all table spaces that do not have custom settings. Custom settings are defined using the **TABLESPACE** *tblspace-name* **ON** *database-alias* clause.

CONTAINER *container-name* **FOR** *tblspace-name* **ON** *database-alias*
Resets the alert settings for the table space container called *container-name*, for the table space specified using the **FOR** *tblspace-name* clause, on the database specified using the **ON** *database-alias* clause. If this table space container has custom settings, then these settings are removed and the current table space containers default is used.

DATABASE ON *database-alias*
Resets the alert settings for the database specified using the **ON** *database-alias* clause. If this database has custom settings, then these settings are removed and the install default is used.

TABLESPACE *tblspace-name* **ON** *database-alias*
Resets the alert settings for the table space called *tblspace-name*, on the database specified using the **ON** *database-alias* clause. If this table space has custom settings, then these settings are removed and the install default is used.

USING *health-indicator-name*
Specifies the set of health indicators for which alert configuration will be reset. Health indicator names consist of a two-letter object identifier followed by a name that describes what the indicator measures. For example:

```
db.sort_privmem_util
```

If you do not specify this option, all health indicators for the specified object or object type will be reset.

Example

Reset alert settings for the database manager that owns the database which contains the ADMIN_CMD procedure.

```
CALL SYSPROC.ADMIN_CMD( 'reset alert cfg for dbm' )
```

Usage notes

Command execution status is returned in the SQLCA resulting from the CALL statement.

The *database-alias* must be a local database defined in the catalog on the server because the ADMIN_CMD procedure runs on the server only.

RESET DATABASE CONFIGURATION command using the ADMIN_CMD procedure

Resets the configuration of a specific database to the system defaults.

Scope

This command only affects the database partition that the application is connected to.

Authorization

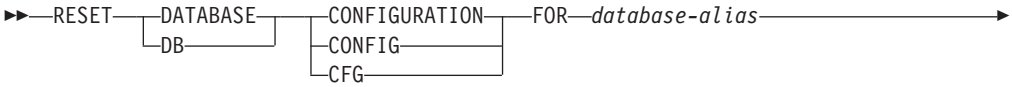
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT

Required connection

Database

Command syntax



Command parameters

FOR *database-alias*
Specifies the alias of the database whose configuration is to be reset to the system defaults. The database alias must be one that is defined in the catalog on the server, and must refer to a local database on the server.

DBPARTITIONNUM *db-partition-num*
If a database configuration reset is to be applied to a specific database partition, this parameter may be used. If this parameter is not provided, the reset will take effect on all database partitions.

Example

Reset the configuration of a database cataloged with alias SAMPLE on the server

```
CALL SYSPROC.ADMIN_CMD( 'reset db cfg for SAMPLE' )
```

Usage notes

To view or print a list of the database configuration parameters, use the SYSIBMADM.DBCFG administration view.

To change the value of a configurable parameter, use the UPDATE DATABASE CONFIGURATION command.

Changes to the database configuration file become effective only after they are loaded into memory. All applications must disconnect from the database before this can occur.

If an error occurs, the database configuration file does not change.

The database configuration file cannot be reset if the checksum is invalid. This might occur if the database configuration file is changed without using the appropriate command. If this happens, the database must be restored to reset the database configuration file.

The RESET DATABASE CONFIGURATION command will reset the database configuration parameters to the documented default configuration values, where **auto_runstats** will be ON. **Self_tuning_mem** will be reset to ON on non-partitioned database environments and to OFF on partitioned database environments.

Command execution status is returned in the SQLCA resulting from the CALL statement.

The *database-alias* must be a local database defined in the catalog on the server because the ADMIN_CMD procedure runs on the server only.

RESET DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure

Resets the parameters in the database manager configuration file to the system defaults for the instance that contains the currently connected database. The values are reset by node type.

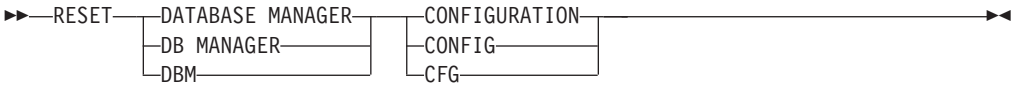
Authorization

SYSADM

Required connection

Database

Command syntax



Command parameters

None

Example

Reset the configuration of the instance which contains the database the ADMIN_CMD stored procedure belongs to.

```
CALL SYSPROC.ADMIN_CMD( 'reset dbm cfg' )
```

Usage notes

This command resets all parameters set by the installation program. This could cause error messages to be returned when restarting DB2. For example, if the **svcename** parameter is reset, the user will receive the SQL5043N error message when trying to restart DB2.

Before running this command, save the output from the SYSIBMADM.DBMCFG administrative view to a file so that you can refer to the existing settings. Individual settings can then be updated using the UPDATE DATABASE MANAGER CONFIGURATION command through the ADMIN_CMD procedure.

It is not recommended that the **svcename** parameter, set by the installation program, be modified by the user.

To view or print a list of the database manager configuration parameters, use the SYSIBMADM.DBMCFG administration view. To change the value of a configurable parameter, use the UPDATE DATABASE MANAGER CONFIGURATION command through the ADMIN_CMD procedure.

For more information about these parameters, refer to the summary list of configuration parameters and the individual parameters.

Some changes to the database manager configuration file become effective only after they are loaded into memory. For more information on which parameters are configurable on-line and which ones are not, see the configuration parameter summary. Server configuration parameters that are not reset immediately are reset during execution of db2start. For a client configuration parameter, parameters are reset the next time you restart the application. If the client is the command line processor, it is necessary to invoke TERMINATE.

If an error occurs, the database manager configuration file does not change.

The database manager configuration file cannot be reset if the checksum is invalid. This might occur if you edit the configuration file manually and do not use the appropriate command. If the checksum is invalid, you must recreate the instance.

REWIND TAPE command using the ADMIN_CMD procedure

Rewinds tapes for backup and restore operations to streaming tape devices. This command is only supported on Windows operating systems.

Authorization

One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT

Required connection

Database

Command syntax

Command parameters

ON *device*
Specifies a valid tape device name. The default value is `\\.\TAPE0`.The device specified must be relative to the server.

Example

Rewind the tape on the device named '\\.\TAPE1'.

```
CALL SYSPROC.ADMIN_CMD( 'rewind tape on \\.\TAPE1' )
```

Usage notes

Command execution status is returned in the SQLCA resulting from the CALL statement.

RUNSTATS command using the ADMIN_CMD procedure

Updates statistics about the characteristics of a table and/or associated indexes, or statistical views. These characteristics include number of records, number of pages, and average record length. The optimizer uses these statistics when determining access paths to the data.

For a table, this utility should be called when the table has had many updates, or after reorganizing the table. For a statistical view, this utility should be called when changes to underlying tables have substantially affected the rows returned by the view. The view must have been previously enabled for use in query optimization using the ALTER VIEW statement.

Scope

This command can be issued from any database partition in the `db2nodes.cfg` file. It can be used to update the catalogs on the catalog database partition.

For tables, this command collects statistics for a table on the database partition from which it is invoked. If the table does not exist on that database partition, the first database partition in the database partition group is selected.

For views, this command collects statistics using data from tables on all participating database partitions.

Authorization

For tables, one of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM
- SQLADM
- CONTROL privilege on the table
- LOAD authority

You do not need any explicit privilege to use this command on any declared temporary table that exists within its connection.

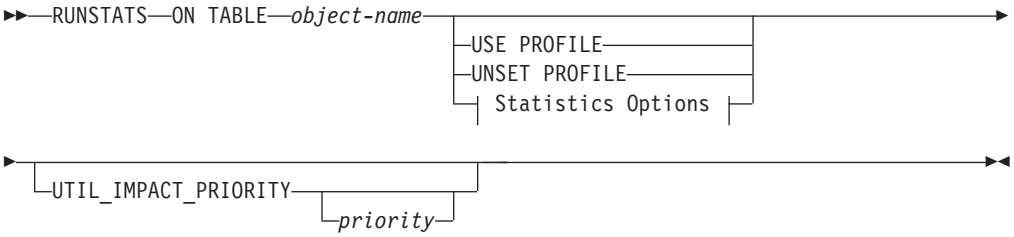
For statistical views, one of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM
- SQLADM
- CONTROL privilege on the statistical view

Required connection

Database

Command syntax



Statistics Options:

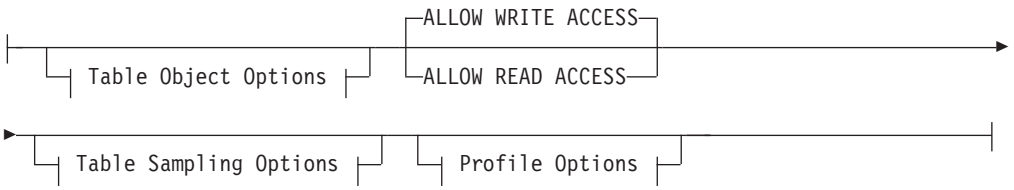


Table Object Options:

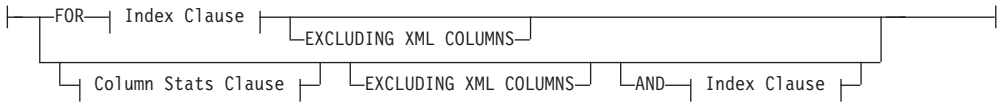
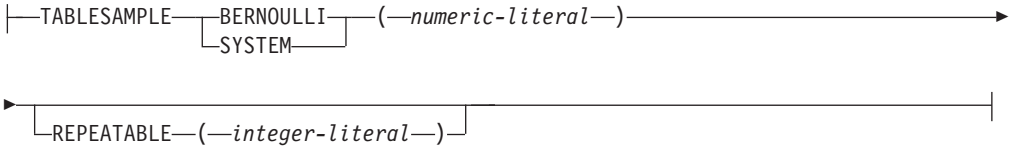
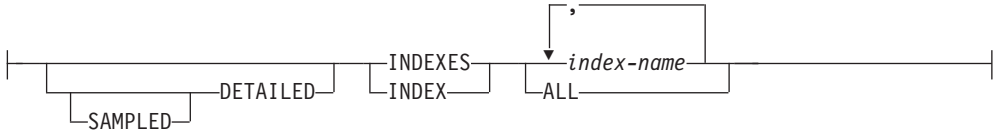


Table Sampling Options:

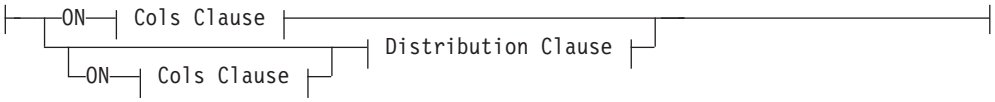


Profile Options:

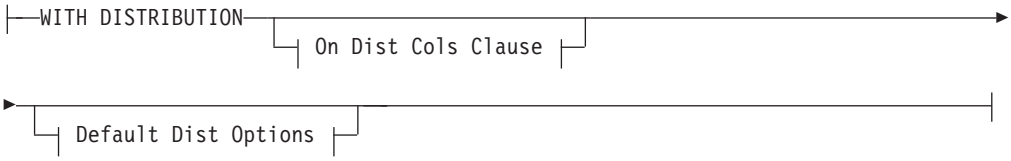
Index Clause:



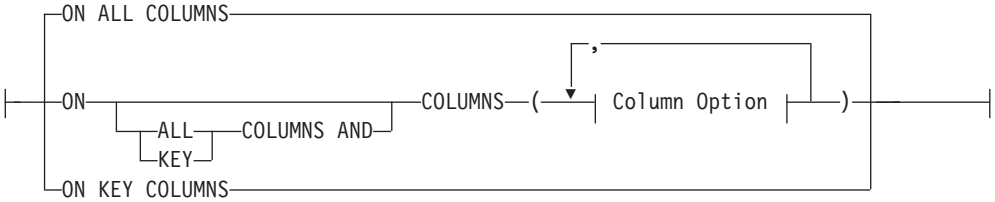
Column Stats Clause:



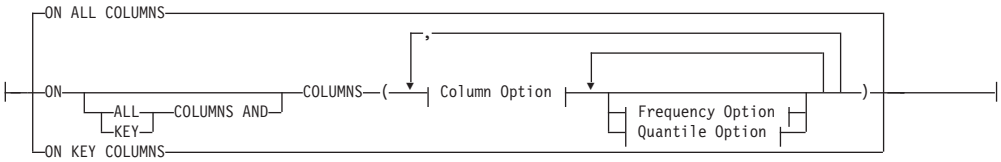
Distribution Clause:



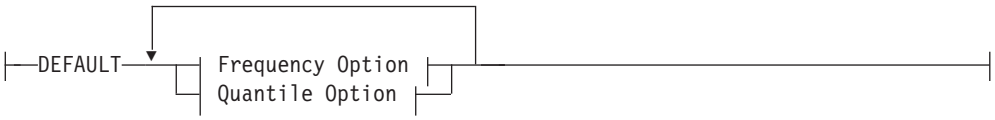
On Cols Clause:



On Dist Cols Clause:



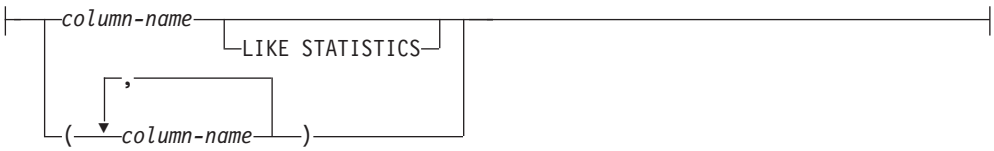
Default Dist Option:



Frequency Option:

Quantile Option:

Column Option:



Command parameters

object-name
Identifies the table or statistical view on which statistics are to be collected. It must not be a hierarchy table. For typed tables, *object-name* must be the name of the root table of the table hierarchy. The fully qualified name or alias in the form: *schema.object-name* must be used. The schema is the user name under which the table was created.

index-name
Identifies an existing index defined on the table. The fully qualified name in the form *schema.index-name* must be used. This option cannot be used for views.

USE PROFILE
This option allows RUNSTATS to employ a previously stored statistics profile to gather statistics for a table or statistical view. The statistics profile is created using the **SET PROFILE** options and is updated using the **UPDATE PROFILE** options.

UNSET PROFILE
Specify this option to remove an existing statistics profile. For example,

```
RUNSTATS ON tablemyschema.mytable UNSET PROFILE
```

FOR INDEXES
Collects and updates statistics for the indexes only. If no table statistics had been previously collected on the table, basic table statistics are also collected. These basic statistics do not include any distribution statistics. This option cannot be used for views.

AND INDEXES
Collects and updates statistics for both the table and the indexes. This option cannot be used for views.

DETAILED
Calculates extended index statistics. These are the `CLUSTERFACTOR` and `PAGE_FETCH_PAIRS` statistics that are gathered for relatively large indexes. This option cannot be used for views.

SAMPLED
This option, when used with the **DETAILED** option, allows RUNSTATS to employ a CPU sampling technique when compiling the extended index statistics. If the option is not specified, every entry in the index is examined to compute the extended index statistics. This option cannot be used for views.

ON ALL COLUMNS
To collect statistics on all eligible columns, use the **ON ALL COLUMNS** clause. Columns can be specified either for basic statistics collection (`On Cols` clause) or in conjunction with the **WITH DISTRIBUTION** clause (`On Dist Cols` clause). The **ON ALL COLUMNS** specification is the default option if neither of the column specific clauses are specified.

If it is specified in the `On Cols` clause, all columns will have only basic column statistics collected unless specific columns are chosen as part of the **WITH DISTRIBUTION** clause. Those columns specified as part of the **WITH DISTRIBUTION** clause will also have basic and distribution statistics collected.

If the **WITH DISTRIBUTION ON ALL COLUMNS** is specified both basic statistics and distribution statistics are collected for all eligible columns. Anything specified in the `On Cols` clause is redundant and therefore not necessary.

ON COLUMNS

This clause allows the user to specify a list of columns for which to collect statistics. If you specify group of columns, the number of distinct values for the group will be collected. When you run RUNSTATS on a table without gathering index statistics, and specify a subset of columns for which statistics are to be gathered, then:

1. Statistics for columns not specified in the RUNSTATS command but which are the first column in an index are NOT reset.
2. Statistics for all other columns not specified in the RUNSTATS command are reset.

This clause can be used in the `On Cols` clause and the `On Dist Cols` clause. Collecting distribution statistics for a group of columns is not currently supported.

If XML type columns are specified in a column group, the XML type columns will be ignored for the purpose of collecting distinct values for the group. However, basic XML column statistics will be collected for the XML type columns in the column group.

EXCLUDING XML COLUMNS

This clause allows you to omit all XML type columns from statistics collection. This clause facilitates the collection of statistics on non-XML columns because the inclusion of XML data can require greater system resources. The **EXCLUDING XML COLUMNS** clause takes precedence over other clauses that specify XML columns for statistics collection. For example, if you use the **EXCLUDING XML COLUMNS** clause, and you also specify XML type columns with the **ON COLUMNS** clause or you use the **ON ALL COLUMNS** clause, all XML type columns will be ignored during statistics collection. For DB2 V9.7 Fix Pack 1 and later releases, distribution statistics over XML type columns are not collected when this clause is specified.

ON KEY COLUMNS

Instead of listing specific columns, you can choose to collect statistics on columns that make up all the indexes defined on the table. It is assumed here that critical columns in queries are also those used to create indexes on the table. If there are no indexes on the table, it is as good as an empty list and no column statistics will be collected. It can be used in the `On Cols` clause or the `On Dist Cols` clause. It is redundant in the `On Cols` clause if specified in both clauses since the **WITH DISTRIBUTION** clause is used to specify collection of both basic and distribution statistics. XML type columns are by definition not a key column and will not be included for statistics collection by the **ON KEY COLUMNS** clause. This option cannot be used for views.

column-name

Name of a column in the table or statistical view. If you specify the name

of an ineligible column for statistics collection, such as a nonexistent column or a mistyped column name, error (-205) is returned. Two lists of columns can be specified, one without distribution and one with distribution. If the column is specified in the list that is not associated with the **WITH DISTRIBUTION** clause only basic column statistics will be collected. If the column appears in both lists, distribution statistics will be collected (unless **NUM_FREQVALUES** and **NUM_QUANTILES** are set to zero).

NUM_FREQVALUES
Defines the maximum number of frequency values to collect. It can be specified for an individual column in the **ON COLUMNS** clause. If the value is not specified for an individual column, the frequency limit value will be picked up from that specified in the **DEFAULT** clause. If it is not specified there either, the maximum number of frequency values to be collected will be what is set in the **num_freqvalues** database configuration parameter.

NUM_QUANTILES
Defines the maximum number of distribution quantile values to collect. It can be specified for an individual column in the **ON COLUMNS** clause. If the value is not specified for an individual column, the quantile limit value will be picked up from that specified in the **DEFAULT** clause. If it is not specified there either, the maximum number of quantile values to be collected will be what is set in the **num_quantiles** database configuration parameter.

For DB2 V9.7 Fix Pack 1 and later releases, distribution statistics for each index over XML data uses a maximum of 250 quantiles as the default. The default can be changed by specifying the **NUM_QUANTILES** parameter in the **ON COLUMNS** or the **DEFAULT** clause. The **num_quantiles** database configuration parameter is ignored while collecting XML distribution statistics.

WITH DISTRIBUTION
This clause specifies that both basic statistics and distribution statistics are to be collected on the columns. If the **ON COLUMNS** clause is not specified, distribution statistics are collected on all the columns of the table or statistical view (excluding columns that are ineligible such as CLOB and LONG VARCHAR). If the **ON COLUMNS** clause is specified, distribution statistics are collected only on the column list provided (excluding those ineligible for statistics collection). If the clause is not specified, only basic statistics are collected.

Collection of distribution statistics on column groups is currently not supported; distribution statistics will not be collected when column groups are specified in the **WITH DISTRIBUTION ON COLUMNS** clause.

DEFAULT
If **NUM_FREQVALUES** or **NUM_QUANTILES** are specified, these values will be used to determine the maximum number of frequency and quantile statistics to be collected for the columns, if these are not specified for individual columns in the **ON COLUMNS** clause. If the **DEFAULT** clause is not specified, the values used will be those in the corresponding database configuration parameters.

LIKE STATISTICS
When this option is specified additional column statistics are collected. These statistics are the `SUB_COUNT` and the `SUB_DELIM_LENGTH` statistics in

SYSSTAT.COLUMNS. The statistics are collected for columns of type CHAR and VARCHAR with a code page attribute of single-byte character set (SBCS), FOR BIT DATA, or UTF-8. They are used by the query optimizer to improve the selectivity estimates for predicates of the type "column LIKE '%xyz'" and "column LIKE '%xyz%'".

ALLOW WRITE ACCESS
Specifies that other users can read from and write to the tables while statistics are calculated. For statistical views, these are the base tables referenced in the view definition.

The **ALLOW WRITE ACCESS** option is not recommended for tables that will have a lot of inserts, updates or deletes occurring concurrently. The RUNSTATS command first performs table statistics and then performs index statistics. Changes in the table's state between the time that the table and index statistics are collected might result in inconsistencies. Although having up-to-date statistics is important for the optimization of queries, it is also important to have consistent statistics. Therefore, statistics should be collected at a time when inserts, updates or deletes are at a minimum.

ALLOW READ ACCESS
Specifies that other users can have read-only access to the tables while statistics are calculated. For statistical views, these are the base tables referenced in the view definition.

TABLESAMPLE BERNOULLI
This option allows RUNSTATS to collect statistics on a sample of the rows from the table or statistical view. *Bernoulli sampling* considers each row individually, including that row with probability *P*/100 (where *P* is the value of numeric-literal) and excluding it with probability 1-*P*/100. Thus, if the numeric-literal were evaluated to be the value 10, representing a 10 percent sample, each row would be included with probability 0.1 and be excluded with probability 0.9. Unless the optional **REPEATABLE** clause is specified, each execution of RUNSTATS will usually yield a different such sample of the table. All data pages will be retrieved through a table scan but only the percentage of rows as specified through the numeric-literal parameter will be used for the statistics collection.

TABLESAMPLE SYSTEM
This option allows RUNSTATS to collect statistics on a sample of the data pages from the tables. *System sampling* considers each page individually, including that page with probability *P*/100 (where *P* is the value of numeric-literal) and excluding it with probability 1-*P*/100. Unless the optional **REPEATABLE** clause is specified, each execution of RUNSTATS will usually yield a different such sample of the table. The size of the sample is controlled by the numeric-literal parameter in parentheses, representing an approximate percentage P of the table to be returned. Only a percentage of the data pages as specified through the numeric-literal parameter will be retrieved and used for the statistics collection.

On statistical views, system sampling is restricted to views whose definitions are a select over a single base table. If the view contains multiple tables, SYSTEM sampling is also possible if:

- the tables are joined using equality predicates on all the primary key and foreign key columns included in a referential integrity constraint defined between the tables,
- no search condition filters rows in any parent tables in the relationship, and

- a single child table, that is also not a parent table, can be identified among all the tables.

If the statistical view does not meet those conditions, Bernoulli sampling will be used instead and a warning will be returned (SQL2317W).

REPEATABLE (*integer-literal***)**
Adding the **REPEATABLE** clause to the **TABLESAMPLE** clause ensures that repeated executions of RUNSTATS return the same sample. The *integer-literal* parameter is a non-negative integer representing the seed to be used in sampling. Passing a negative seed will result in an error (SQL1197N). The sample set might still vary between repeatable RUNSTATS invocations if activity against the table or statistical view resulted in changes to the table or statistical view data since the last time **TABLESAMPLE REPEATABLE** was run. Also, the method by which the sample was obtained as specified by the **BERNOULLI** or **SYSTEM** keyword, must also be the same to ensure consistent results.

numeric-literal
The numeric-literal parameter specifies the size of the sample to be obtained, as a percentage *P*. This value must be a positive number that is less than or equal to 100, and can be between 1 and 0. For example, a value of 0.01 represents one one-hundredth of a percent, such that 1 row in 10,000 would be sampled, on average. A value of 0 or 100 will be treated by the DB2 database system as if sampling was not specified, regardless of whether **TABLESAMPLE BERNOULLI** or **TABLESAMPLE SYSTEM** is specified. A value greater than 100 or less than 0 will be treated as an error (SQL1197N) by the DB2 database system.

SET PROFILE NONE
Specifies that no statistics profile will be set for this RUNSTATS invocation.

SET PROFILE
Allows RUNSTATS to generate and store a specific statistics profile in the system catalog tables and executes the RUNSTATS command options to gather statistics.

SET PROFILE ONLY
Allows RUNSTATS to generate and store a specific statistics profile in the system catalog tables without running the RUNSTATS command options.

UPDATE PROFILE
Allows RUNSTATS to modify an existing statistics profile in the system catalog tables, and runs the RUNSTATS command options of the updated statistics profile to gather statistics. You cannot use the UPDATE PROFILE option to remove clauses that are in a statistics profile.

UPDATE PROFILE ONLY
Allows RUNSTATS to modify an existing statistics profile in the system catalog tables without running the RUNSTATS command options of the updated statistics profile. You cannot use the UPDATE PROFILE ONLY option to remove clauses that are in a statistics profile.

UTIL_IMPACT_PRIORITY *priority*
Specifies that RUNSTATS will be throttled at the level specified by *priority*. *priority* is a number in the range of 1 to 100, with 100 representing the highest priority and 1 representing the lowest. The priority specifies the amount of throttling to which the utility is subjected. All utilities at the same priority undergo the same amount of throttling, and utilities at lower priorities are throttled more than those at higher priorities. If *priority* is not

specified, the RUNSTATS will have the default priority of 50. Omitting the **UTIL_IMPACT_PRIORITY** keyword will invoke the RUNSTATS utility without throttling support. If the **UTIL_IMPACT_PRIORITY** keyword is specified, but the **util_impact_lim** configuration parameter is set to 100, then the utility will run unthrottled. This option cannot be used for views.

In a partitioned database, when used on tables, the RUNSTATS command collects the statistics on only a single database partition. If the database partition from which the RUNSTATS command is executed has a partition of the table, then the command executes on that database partition. Otherwise, the command executes on the first database partition in the database partition group across which the table is partitioned.

Example

Collect statistics on all columns used in indexes and on all indexes.

```
CALL SYSPROC.ADMIN_CMD ('RUNSTATS ON TABLE db2user.employee
  ON KEY COLUMNS and INDEXES ALL')
```

Usage notes

1. When there are detached partitions on a partitioned table, index keys that still belong to detached data partitions which require cleanup will not be counted as part of the keys in the statistics. These keys are not counted because they are invisible and no longer part of the table. They will eventually get removed from the index by asynchronous index cleanup. As a result, statistics collected before asynchronous index cleanup is run will be misleading. If the RUNSTATS command is issued before asynchronous index cleanup completes, it will likely generate a false alarm for index reorganization or index cleanup based on the inaccurate statistics. Once asynchronous index cleanup is run, all the index keys that still belong to detached data partitions which require cleanup will be removed and this may eliminate the need for index reorganization.

   For partitioned tables, you are encouraged to issue the RUNSTATS command after an asynchronous index cleanup has completed in order to generate accurate index statistics in the presence of detached data partitions. To determine whether or not there are detached data partitions in the table, you can check the status field in the SYSCAT.DATAPARTITIONS catalog view and look for the value `L` (logically detached), `I` (index cleanup), or `D` (detached with dependent MQT).

   The RUNSTATS command collects statistics for all index partitions of a partitioned index. Statistics in the SYSTAT.INDEXES view for the partitioned index represent an index partition, except for FIRSTKEYCARD, FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, and FULLKEYCARD statistics. Because these statistics are used in cardinality estimates, they are for the entire index and not for an index partition. Distribution statistics (frequent values and quantiles) are not collected for partitioned indexes, but are gathered if RUNSTATS is run on the table. Statistics on the leading columns of a partitioned index might not be as accurate as statistics on the leading columns of a nonpartitioned index.
2. Command execution status is returned in the SQLCA resulting from the CALL statement.
3. It is recommended to run the RUNSTATS command:

- On tables that have been modified considerably (for example, if a large number of updates have been made, or if a significant amount of data has been inserted or deleted or if LOAD has been done without the statistics option during LOAD).
- On tables that have been reorganized (using REORG, REDISTRIBUTE DATABASE PARTITION GROUP).
- On tables which have been row compressed.
- When a new index has been created.
- Before binding applications whose performance is critical.
- When the prefetch quantity is changed.
- On statistical views whose underlying tables have been modified substantially so as to change the rows that are returned by the view.
- After LOAD has been executed with the **STATISTICS** option, use the RUNSTATS utility to collect statistics on XML columns. Statistics for XML columns are never collected during LOAD, even when LOAD is executed with the **STATISTICS** option. When RUNSTATS is used to collect statistics for XML columns only, existing statistics for non-XML columns that have been collected by LOAD or a previous execution of the RUNSTATS utility are retained. In the case where statistics on some XML columns have been collected previously, the previously collected statistics for an XML column will either be dropped if no statistics on that XML column are collected by the current command, or be replaced if statistics on that XML column are collected by the current command.

4. The options chosen must depend on the specific table and the application. In general:
   - If the table is a very critical table in critical queries, is relatively small, or does not change too much and there is not too much activity on the system itself, it might be worth spending the effort on collecting statistics in as much detail as possible.
   - If the time to collect statistics is limited, if the table is relatively large, or if the table is updated frequently, it might be beneficial to execute RUNSTATS limited to the set of columns that are used in predicates. This way, you will be able to execute the RUNSTATS command more often.
   - If time to collect statistics is very limited and the effort to tailor the RUNSTATS command on a table by table basis is a major issue, consider collecting statistics for the "KEY" columns only. It is assumed that the index contains the set of columns that are critical to the table and are most likely to appear in predicates.
   - If time to collect statistics is very limited and table statistics are to be gathered, consider using the **TABLESAMPLE** option to collect statistics on a subset of the table data.
   - If there are many indexes on the table and **DETAILED** (extended) information on the indexes might improve access plans, consider the **SAMPLED** option to reduce the time it takes to collect statistics.
   - If there is skew in certain columns and predicates of the type `"column = constant"`, it might be beneficial to specify a larger **NUM_FREQVALUES** value for that column
   - Collect distribution statistics for all columns that are used in equality predicates and for which the distribution of values might be skewed.

- For columns that have range predicates (for example "column >= constant", "column BETWEEN constant1 AND constant2") or of the type "column LIKE '%xyz'", it might be beneficial to specify a larger **NUM_QUANTILES** value.
- If storage space is a concern and one cannot afford too much time on collecting statistics, do not specify high **NUM_FREQVALUES** or **NUM_QUANTILES** values for columns that are not used in predicates.
- If index statistics are requested, and statistics have never been run on the table containing the index, statistics on both the table and indexes are calculated.
- If statistics for XML columns in the table are not required, the **EXCLUDING XML COLUMNS** option can be used to exclude all XML columns. This option takes precedence over all other clauses that specify XML columns for statistics collection.

5. After the command is run note the following:
   - A COMMIT should be issued to release the locks.
   - To allow new access plans to be generated, the packages that reference the target table must be rebound.
   - Executing the command on portions of the table could result in inconsistencies as a result of activity on the table since the command was last issued. In this case a warning message is returned. Issuing RUNSTATS on the table only might make table and index level statistics inconsistent. For example, you might collect index level statistics on a table and later delete a significant number of rows from the table. If you then issue RUNSTATS on the table only, the table cardinality might be less than FIRSTKEYCARD, which is an inconsistency. In the same way, if you collect statistics on a new index when you create it, the table level statistics might be inconsistent.
6. The RUNSTATS command will drop previously collected distribution statistics if table statistics are requested. For example, RUNSTATS ON TABLE, or RUNSTATS ON TABLE ... AND INDEXES ALL will cause previously collected distribution statistics to be dropped. If the command is run on indexes only then previously collected distribution statistics are retained. For example, RUNSTATS ON TABLE ... FOR INDEXES ALL will cause the previously collected distribution statistics to be retained. If the RUNSTATS command is run on XML columns only, then previously collected basic column statistics and distribution statistics are retained. In the case where statistics on some XML columns have been collected previously, the previously collected statistics for an XML column will either be dropped if no statistics on that XML column are collected by the current command, or be replaced if statistics on that XML column are collected by the current command.
7. For DB2 V9.7 Fix Pack 1 and later releases, distribution statistics are collected on indexes over XML data defined on an XML column. When the RUNSTATS command is run on a table with the **WITH DISTRIBUTION** clause, the following apply to the collection of distribution statistics on a column of type XML:
   - Distribution statistics are collected for each index over XML data specified on an XML column.
   - The RUNSTATS command must collect both distribution statistics and table statistics to collect distribution statistics for indexes over XML data defined on an XML column. Table statistics must be gathered in order for distribution statistics to be collected since XML distribution statistics are stored with table statistics.

An index clause is not required to collect XML distribution statistics. Specifying only an index clause does not collect XML distribution statistics

By default, XML distribution statistics use a maximum of 250 quantiles for each index over XML data. When collecting distribution statistics on an XML column, you can change the maximum number of quantiles by specifying a value with **NUM_QUANTILES** parameter in the **ON COLUMNS** or the **DEFAULT** clause.

- Distribution statistics are collected for indexes over XML data of type VARCHAR, DOUBLE, TIMESTAMP, and DATE. Distribution statistics are not collected over indexes of type VARCHAR HASHED.
- Distribution statistics are not collected for partitioned indexes over XML data defined on a partitioned table.

8. For range-clustered tables, there is a special system-generated index in the catalog tables which represents the range ordering property of range-clustered tables. When statistics are collected on this type of table, if the table is to be included as part of the statistics collection, statistics will also be collected for the system-generated index. The statistics reflect the fast access of the range lookups by representing the index as a two-level index with as many pages as the base data table, and having the base data clustered perfectly along the index order.
9. In the `On Dist Cols` clause of the command syntax, the Frequency Option and Quantile Option parameters are currently not supported for column `GROUPS`. These options are supported for single columns.
10. There are three prefetch statistics that cannot be computed when working in DMS mode. When looking at the index statistics in the index catalogs, you will see a `-1` value for the following statistics:
    - `AVERAGE_SEQUENCE_FETCH_PAGES`
    - `AVERAGE_SEQUENCE_FETCH_GAP`
    - `AVERAGE_RANDOM_FETCH_PAGES`
11. RUNSTATS sampling through **TABLESAMPLE** only occurs with table data pages and not index pages. When index statistics as well as sampling is requested, all the index pages are scanned for statistics collection. It is only in the collection of table statistics where **TABLESAMPLE** is applicable. However, a more efficient collection of detailed index statistics is available through the **SAMPLED DETAILED** option. This is a different method of sampling than that employed by **TABLESAMPLE** and only applies to the detailed set of index statistics.
12. A statistics profile can be set or updated for the table or statistical view specified in the RUNSTATS command, by using the set profile or update profile options. The statistics profile is stored in a visible string format, which represents the RUNSTATS command, in the `STATISTICS_PROFILE` column of the `SYSCAT.TABLES` system catalog table.
13. Statistics collection on XML type columns is governed by two DB2 database system registry values: **DB2_XML_RUNSTATS_PATHID_K** and **DB2_XML_RUNSTATS_PATHVALUE_K**. These two parameters are similar to the **NUM_FREQVALUES** parameter in that they specify the number of frequency values to collect. If not set, a default of 200 will be used for both parameters.
14. RUNSTATS acquires an IX table lock on SYSTABLES and a U lock on the row for the table on which statistics are being gathered at the beginning of RUNSTATS. Operations can still read from SYSTABLES including the row with the U lock. Write operations are also possible, providing they do not

occur against the row with the U lock. However, another reader or writer will not be able acquire an S lock on SYSTABLES because of RUNSTATS' IX lock.

15. Statistics are not collected for columns with structured types. If they are specified, columns with these data types are ignored.
16. Only AVGCOLLEN and NUMNULLS are collected for columns with LOB or LONG data types.
17. AVGCOLLEN represents the average space in bytes when the column is stored in database memory or a temporary table. This value represents the length of the data descriptor for LOB or LONG data types, except when LOB data is inlined on the data page.

    Note: The average space required to store the column on disk may be different than the value represented by this statistic.

SET TAPE POSITION command using the ADMIN_CMD procedure

Sets the positions of tapes for backup and restore operations to streaming tape devices. This command is only supported on Windows operating systems.

Authorization

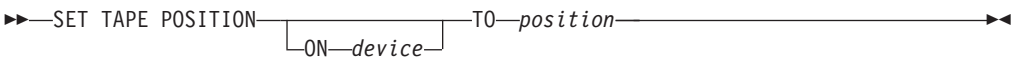
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT

Required connection

Database

Command syntax



Command parameters

ON *device*
Specifies a valid tape device name. The default value is \\.\TAPE0. The device specified must be relative to the server.

TO *position*
Specifies the mark at which the tape is to be positioned. DB2 for Windows writes a tape mark after every backup image. A value of 1 specifies the first position, 2 specifies the second position, and so on. If the tape is positioned at tape mark 1, for example, archive 2 is positioned to be restored.

Example

Because DB2 databases write a tape mark after every backup image, specifying a position of 1 will move the tape to the start of the second archive on the tape.

```
CALL SYSPROC.ADMIN_CMD( 'set tape position to 1' )
```

Usage notes

Command execution status is returned in the SQLCA resulting from the CALL statement.

UNQUIESCE DATABASE command using the ADMIN_CMD procedure

Restores user access to databases which have been quiesced for maintenance or other reasons.UNQUIESCE restores user access without necessitating a shutdown and database restart.

Unless specifically designated, no user except those with SYSADM, SYSMAINT, or SYSCTRL has access to a database while it is quiesced. Therefore an UNQUIESCE is required to restore general access to a quiesced database.

Scope

UNQUIESCE DB restores user access to all objects in the quiesced database.

To stop the instance and unquiesce it and all its databases, issue the db2stop command. Stopping and restarting DB2 will unquiesce all instances and databases.

Authorization

One of the following:

For database level unquiesce:

- SYSADM
- DBADM

Command syntax

Required connection

Database

Command parameters

DB Unquiesce the database. User access will be restored to all objects in the database.

Examples

Unquiescing a Database

```
CALL SYSPROC.ADMIN_CMD( 'unquiesce db' )
```

This command will unquiesce the database that had previously been quiesced.

Usage notes

Command execution status is returned in the SQLCA resulting from the CALL statement.

UPDATE ALERT CONFIGURATION command using the ADMIN_CMD procedure

Updates the alert configuration settings for health indicators.

Important: This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Authorization

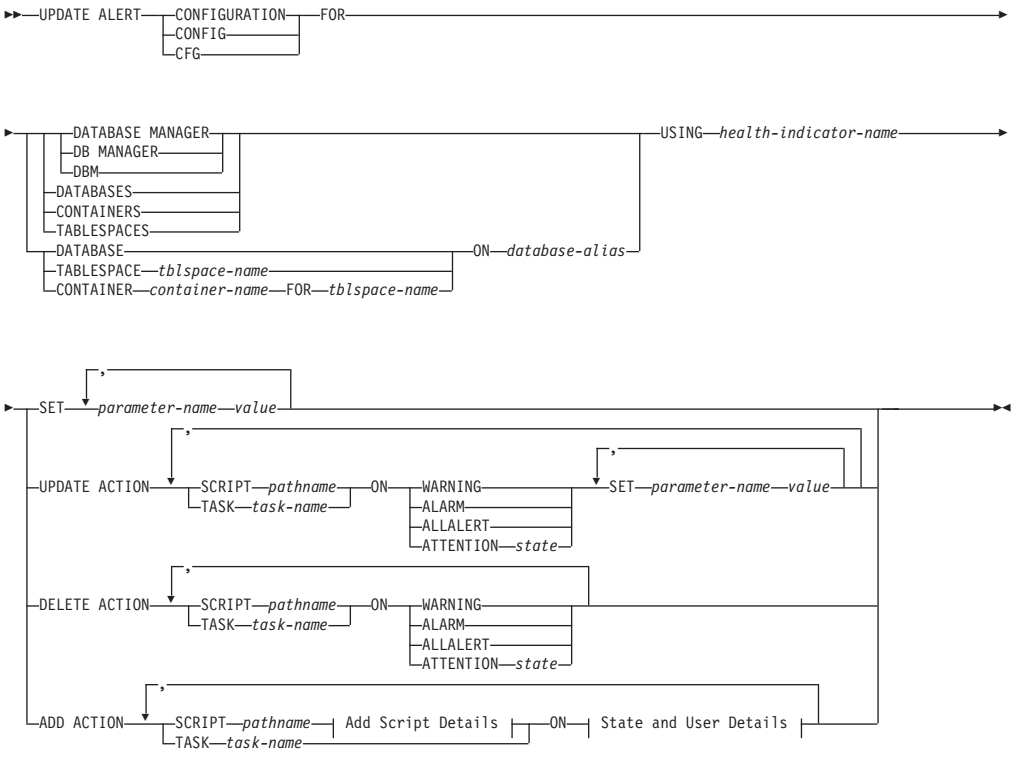
One of the following:

- SYSADM
- SYSMAINT
- SYSCTRL

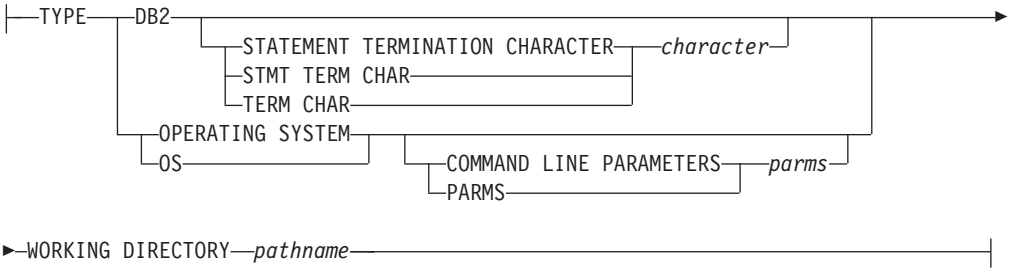
Required Connection

Database

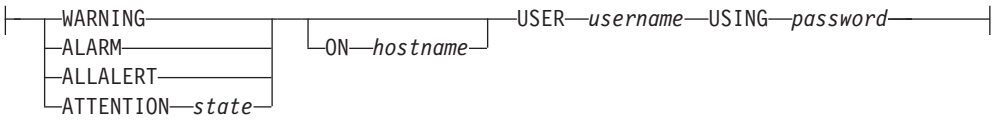
Command Syntax



Add Script Details:



State and User Details:



Command Parameters

DATABASE MANAGER
Updates alert settings for the database manager.

DATABASES
Updates alert settings for all databases managed by the database manager. These are the settings that apply to all databases that do not have custom settings. Custom settings are defined using the **DATABASE ON** *database-alias* clause.

CONTAINERS
Updates alert settings for all table space containers managed by the database manager. These are the settings that apply to all table space containers that do not have custom settings. Custom settings are defined using the **CONTAINER** *container-name* **ON** *database-alias* clause.

TABLESPACES
Updates alert settings for all table spaces managed by the database manager. These are the settings that apply to all table spaces that do not have custom settings. Custom settings are defined using the **TABLESPACE** *tblspace-name* **ON** *database-alias* clause.

DATABASE ON *database-alias*
Updates the alert settings for the database specified using the **ON** *database-alias* clause. If this database has custom settings, then they override the settings for all databases for the instance, which is specified using the **DATABASES** parameter.

CONTAINER *container-name* **FOR** *tblspace-name* **ON** *database-alias*
Updates the alert settings for the table space container called *container-name*, for the table space specified using the **FOR** *tblspace-name* clause, on the database specified using the **ON** *database-alias* clause. If this table space container has custom settings, then they override the settings for all table space containers for the database, which is specified using the **CONTAINERS** parameter.

TABLESPACE *tblspace-name* **ON** *database-alias*
Updates the alert settings for the table space called *name*, on the database specified using the **ON** *database-alias* clause. If this table space has custom settings, then they override the settings for all table spaces for the database, which is specified using the **TABLESPACES** parameter.

USING *health-indicator-name*

Specifies the set of health indicators for which alert configuration will be updated. Health indicator names consist of a two-letter object identifier followed by a name which describes what the indicator measures. For example:

```
db.sort_privmem_util
```

SET *parameter-name value*

Updates the alert configuration element, *parameter-name*, of the health indicator to the specified value. *parameter-name* must be one of the following:

- ALARM: the *value* is a health indicator unit.
- WARNING: the *value* is a health indicator unit.
- SENSITIVITY: the *value* is in seconds.
- ACTIONSENABLED: the *value* can be either YES or NO.
- THRESHOLDSCHECKED: the *value* can be either YES or NO.

The list of possible health indicator units for your specific DB2 version can be gathered by running the following query :

```
SELECT SUBSTR(UNIT,1,80) AS UNIT
   FROM TABLE(HEALTH_GET_IND_DEFINITION('')) AS T GROUP BY UNIT
```

UPDATE ACTION SCRIPT *pathname* **ON [WARNING | ALARM | ALLALERT | ATTENTION** *state***]**

Specifies that the script attributes of the predefined script with absolute path name *pathname* will be updated according to the following clause:

SET *parameter-name value*

Updates the script attribute, *parameter-name*, to the specified value. *parameter-name* must be one of the following:

- SCRIPTTYPE

  OS or DB2 are the valid types.
- WORKINGDIR
- TERMCHAR
- CMDLINEPARMS

  The command line parameters that you specify for the operating system script will precede the default supplied parameters . The parameters that are sent to the operating system script are:
  - List of user supplied parameters
  - Health indicator short name
  - Fully qualified object name
  - Health indicator value
  - Alert state
- USERID
- PASSWORD
- SYSTEM

UPDATE ACTION TASK *task-name* **ON [WARNING | ALARM | ALLALERT | ATTENTION** *state***]**

Specifies that the task attributes of the task with name *name* will be updated according to the following clause:

SET *parameter-name value*

Updates the task attribute, *parameter-name*, to the specified value. *parameter-name* must be one of the following:

- USERID
- PASSWORD
- SYSTEM

DELETE ACTION SCRIPT *pathname* **ON [WARNING | ALARM | ALLALERT | ATTENTION** *state*]
Removes the action script with absolute path name *pathname* from the list of alert action scripts.

DELETE ACTION TASK *task-name* **ON [WARNING | ALARM | ALLALERT | ATTENTION** *state*]
Removes the action task called *name* from the list of alert action tasks.

ADD ACTION SCRIPT *pathname* **ON [WARNING | ALARM | ALLALERT | ATTENTION** *state*]
Specifies that a new action script with absolute path name *pathname* is to be added, the attributes of which are given by the following:

TYPE An action script must be either a DB2 Command script or an operating system script:
- DB2
- OPERATING SYSTEM

If it is a DB2 Command script, then the following clause allows one to optionally specify the character, *character*, that is used in the script to terminate statements:

```
STATEMENT TERMINATION CHARACTER ;
```

If it is an operating system script, then the following clause allows one to optionally specify the command-line parameters, *parms*, that would be passed to the script upon invocation: **COMMAND LINE PARAMETERS** *parms*

WORKING DIRECTORY *pathname*
Specifies the absolute path name, *pathname*, of the directory in which the script will be executed.

USER *username* **USING** *password*
Specifies the user account, *username*, and associated password, *password*, under which the script will be executed. When using the **ADD ACTION** option, the *username* and *password* might be exposed in the network (where the *username* and *password* are sent unencrypted), the db2diag log file, trace files, dump file, snapshot monitor (dynamic SQL snapshot), system monitor snapshots, a number of event monitors (such as statement, deadlock), Query Patroller, explain tables, db2pd output (such as package cache and lock timeout mechanisms) and DB2 audit records.

ADD ACTION TASK *name* **ON [WARNING | ALARM | ALLALERT | ATTENTION** *state*]
Specifies that a new task, called *name*, is to be added to be run **ON** the specified condition.

ON [WARNING | ALARM | ALLALERT | ATTENTION *state*]
Specifies the condition on which the action or task will run. For threshold-based health indicators (HIs), this is **WARNING** or **ALARM**. For state-based HIs, this can be a numeric state as documented for each state-based HI (for example, for the ts.ts_op_status health indicator, refer to the **tablespace_state** monitor element for table space states), or a text

identifier for this state. **ALLALERTS** handles any changes in the state for threshold-based HIs and state-based HIs (for example, the state changes from warning to normal).

ATTENTION *state*

Valid numerical values for some of the database health indicator states are given below as an example for the ADD ACTION SCRIPT CLP command option:

- 0 - Active; Normal (ACTIVE)
- 1 - Quiesce pending (QUIESCE_PEND)
- 2 - Quiesced (QUIESCED)
- 3 - Rollforward (ROLLFWD)

Additional state-based health indicators are defined in the header files `sqlmon.h` and `sqlutil.h`.

The UPDATE ALERT CFG command called by the ADMIN_CMD stored procedure supports either a numeric value or a text identifier for *state*. Valid numerical values and text identifiers for some additional health indicator states, as an example for the table space operational status health indicator (ts.ts_op_status), are:

- 0x1 - QUIESCED_SHARE
- 0x2 - QUIESCED_UPDATE
- 0x4 - QUIESCED_EXCLUSIVE

Using the UPDATE ALERT CFG command and the above health indicator values, the following command line entry,

```
ADD ACTION SCRIPT ... ON ATTENTION 2
```

is equivalent to

```
ADD ACTION SCRIPT ... ON ATTENTION QUIESCED_UPDATE
```

In addition, for the table space operational status health indicator (ts.ts_op_status), you can specify multiple states using a single numeric value by OR'ing states together. For example, you can specify state 7 (= 0x1 + 0x2 + 0x4), the action will be performed when the table space enters any of the Quiesced: SHARE, Quiesced: UPDATE or Quiesce: EXCLUSIVE states. Alternatively, you could specify QUIESCED_SHARE, QUIESCED_UPDATE, and QUIESCED_EXCLUSIVE in three separate UPDATE ALERT CFG command executions.

Example

Add an action for the db.log_fs_util indicator that will execute the script `/home/test/scripts/logfsutilact` when there is an alarm on the system with hostname 'plato'.

```
CALL SYSPROC.ADMIN_CMD( 'update alert cfg for databases using
   db.log_fs_util add action script /home/test/scripts/logfsutilact
   type os command line parameters "param1 param2" working
   directory /tmp on alarm on plato user dricard using mypasswdv' )
```

To check the alert configuration after it has been set, you can use the HEALTH_GET_IND_DEFINITION and HEALTH_GET_ALERT_ACTION_CFG table functions as follows:

```
SELECT OBJECTTYPE, ID, CONDITION, ACTIONTYPE,
   SUBSTR(ACTIONNAME,1,50) AS ACTION_NAME
   FROM TABLE(SYSPROC.HEALTH_GET_ALERT_ACTION_CFG('DB','G','',''))
   AS ALERT_ACTION_CFG
```

The following is an example of output from this query:

```
OBJECTTYPE ID     CONDITION  ACTIONTYPE ACTION_NAME
---------- ------ ---------- ---------- --------------------------------
DB           1006 ALARM      S          /home/dricard/scripts/logfsutilact

   1 record(s) selected.
```

Usage notes

For the ADD ACTION option, the supplied *username* and *password* may be exposed in various places where SQL statement text is captured:

- the network (username/password are passed over the wire unencrypted)
- db2diag log file
- trace files
- dump file
- snapshot monitor (dynamic SQL snapshot)
- system monitor snapshots
- a number of event monitors (statement, deadlock)
- query patroller
- explain tables
- db2pd output (package cache and lock timeout mechanisms, among others)
- DB2 audit records

Command execution status is returned in the SQLCA resulting from the CALL statement.

The *database-alias* must be defined in the catalog on the server and be local to the server.

The *pathname* must be with a fully-qualified server path name.

UPDATE CONTACT command using the ADMIN_CMD procedure

Updates the attributes of a contact that is defined on the local system. A contact is a user to whom the Scheduler and Health Monitor send messages.

To create a contact, use the ADD CONTACT command. The setting of the Database Administration Server (DAS) **contact_host** configuration parameter determines whether the list is local or global.

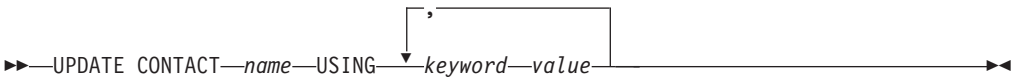
Authorization

None

Required connection

Database. The DAS must be running.

Command syntax



Command parameters

UPDATE CONTACT *name*
The name of the contact that will be updated.

USING *keyword value*
Specifies the contact parameter to be updated (*keyword*) and the value to which it will be set (*value*). The valid set of keywords is:

ADDRESS
The e-mail address that is used by the SMTP server to send the notification.

TYPE Whether the address is for an e-mail address or a pager.

MAXPAGELEN
The maximum number of characters that the pager can accept.

DESCRIPTION
A textual description of the contact. This has a maximum length of 128 characters.

Example

Update the address of user 'test' to 'newaddress@test.com'.

```
CALL SYSPROC.ADMIN_CMD( 'update contact test using address newaddress@test.com' )
```

Usage notes

The DAS must have been created and be running.

Command execution status is returned in the SQLCA resulting from the CALL statement.

UPDATE CONTACTGROUP command using the ADMIN_CMD procedure

Updates the attributes of a contact group that is defined on the local system. A contact group is a list of users who should be notified by the Scheduler and the Health Monitor.

The setting of the Database Administration Server (DAS) **contact_host** configuration parameter determines whether the list is local or global.

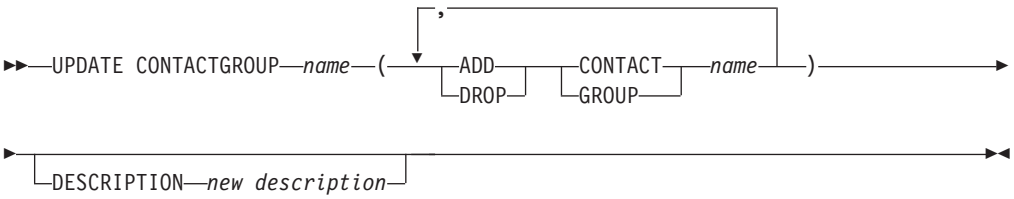
Authorization

None

Required Connection

Database. The DAS must be running.

Command Syntax



Command Parameters

CONTACTGROUP *name*
Name of the contact group which will be updated.

ADD CONTACT *name*
Specifies the name of the new contact to be added to the group. A contact can be defined with the ADD CONTACT command after it has been added to a group.

DROP CONTACT *name*
Specifies the name of a contact in the group that will be dropped from the group.

ADD GROUP *name*
Specifies the name of the new contact group to be added to the group.

DROP GROUP *name*
Specifies the name of a contact group that will be dropped from the group.

DESCRIPTION *new description*
Optional. A new textual description for the contact group.

Example

Add the contact named 'cname2' to the contact group named 'gname1':

```
CALL SYSPROC.ADMIN_CMD( 'update contactgroup gname1 add contact cname2' )
```

Usage notes

The DAS must have been created and be running.

Command execution status is returned in the SQLCA resulting from the CALL statement.

UPDATE DATABASE CONFIGURATION command using the ADMIN_CMD procedure

Modifies individual entries in a specific database configuration file. A database configuration file resides on every database partition on which the database has been created.

Scope

This command updates all database partitions by default, except when DBPARTITIONNUM is specified to update only one database partition.

Authorization

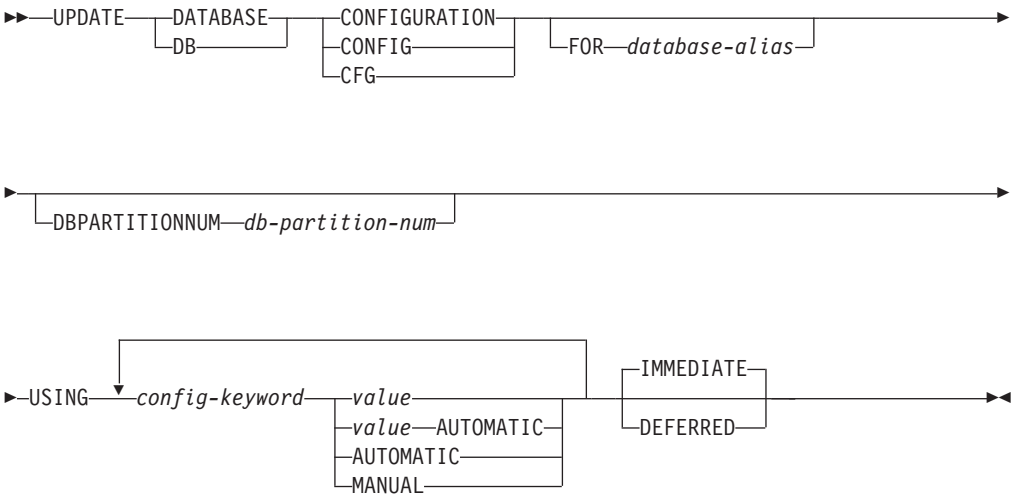
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT

Required connection

Database. The database connection must be local to the instance containing the connected database.

Command syntax



Command parameters

AUTOMATIC

Some configuration parameters can be set to AUTOMATIC, allowing DB2 to automatically adjust these parameters to reflect the current resource requirements. For a list of configuration parameters that support the **AUTOMATIC** keyword, refer to the configuration parameters summary. If a value is specified along with the **AUTOMATIC** keyword, it might influence the automatic calculations. For specific details about this behavior, refer to the documentation for the configuration parameter.

Note: The **appl_memory**, **logindexbuild**, **max_log** and **num_log_span** database configuration parameters can only be set to AUTOMATIC using the command line processor.

DEFERRED

Make the changes only in the configuration file, so that the changes take effect the next time you reactivate the database.

FOR *database-alias*

Specifies the alias of the database whose configuration is to be updated. Specifying the database alias is not required when a database connection has already been established. The database alias must be defined locally on the server. You can update the configuration file for another database

residing under the same database instance. For example, if you are connected only to database `db11`, and issue `update db config for alias db22 using .... immediate`:

- If there is no active connection on `db22`, the update will be successful because only the configuration file needs to be updated. A new connection (which will activate the database) will see the new change in memory.
- If there are active connections on `db22` from other applications, the update will work on disk but not in memory. You will receive a warning saying that the database needs to be restarted.

DBPARTITIONNUM *db-partition-num*
If a database configuration update is to be applied to a specific database partition, this parameter may be used. If this parameter is not provided, the update will take effect on all database partitions.

IMMEDIATE
Make the changes immediately, while the database is running. **IMMEDIATE** is the default action. Since the ADMIN_CMD procedure requires a database connection, the changes will be effective immediately for any dynamically configurable parameters for the connected database.

This is a default clause when operating in the CLPPlus interface as well. **IMMEDIATE** need not be called when using CLPPlus processor.

MANUAL
Disables automatic tuning for the configuration parameter. The parameter is set to its current internal value and is no longer updated automatically.

USING *config-keyword value*
config-keyword specifies the database configuration parameter to be updated. *value* specifies the value to be assigned to the parameter.

Example

Set the database configuration parameter **sortheap** to a value of 1000 on the database partition to which the application is currently connected to.

```
CALL SYSPROC.ADMIN_CMD ('UPDATE DB CFG USING sortheap 1000')
```

Usage notes

Command execution status is returned in the SQLCA resulting from the CALL statement.

The *database-alias* must be an alias name that is defined on the server.

The command affect all database partitions unless **DBPARTITIONNUM** is specified.

To view or print a list of the database configuration parameters, use the SYSIBMADM.DBCFG administration view.

To reset all the database configuration parameters to the recommended defaults, use the RESET DATABASE CONFIGURATION command using the ADMIN_CMD procedure.

To change a database configuration parameter, use the UPDATE DATABASE CONFIGURATION command through the ADMIN_CMD procedure. For example,

to change the logging mode to “archival logging” on a single-partition database environment containing a database called ZELLMART, use:

```
CALL SYSPROC.ADMIN_CMD ('update db cfg for zellmart using logretain recovery')
```

To check that the **logretain** configuration parameter has changed, use:

```
SELECT * FROM SYSIBMADM.DBCFG WHERE NAME='logretain'
```

To update a database configuration parameter on a specific database partition, you can:

1. set the **DB2NODE** variable to a database partition number.
2. connect to the database partition.
3. update the database configuration parameters using UPDATE DATABASE CONFIGURATION command through the ADMIN_CMD procedure.
4. disconnect from the database partition.

or you can use **DBPARTITIONNUM**. For example, to update the logging mode to only one specific partition (30) using DBPARTITIONNUM, use:

```
CALL SYSPROC.ADMIN_CMD ('update db cfg for zellmart dbpartitionnum 30 using
 logretain recovery')
```

For more information about DB2 configuration parameters and the values available for each type of database node, see the individual configuration parameter descriptions. The values of these parameters differ for each type of database node configured (server, client, or server with remote clients).

Not all parameters can be updated.

Some changes to the database configuration file become effective only after they are loaded into memory. All applications must disconnect from the database before this can occur. For more information on which parameters are configurable online and which ones are not, see summary list of configuration parameters.

If an error occurs, the database configuration file does not change. The database configuration file cannot be updated if the checksum is invalid. This might occur if the database configuration file is changed without using the appropriate command. If this happens, the database must be restored to reset the database configuration file.

UPDATE DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure

Modifies individual entries in the database manager configuration file for the instance that contains the currently connected database.

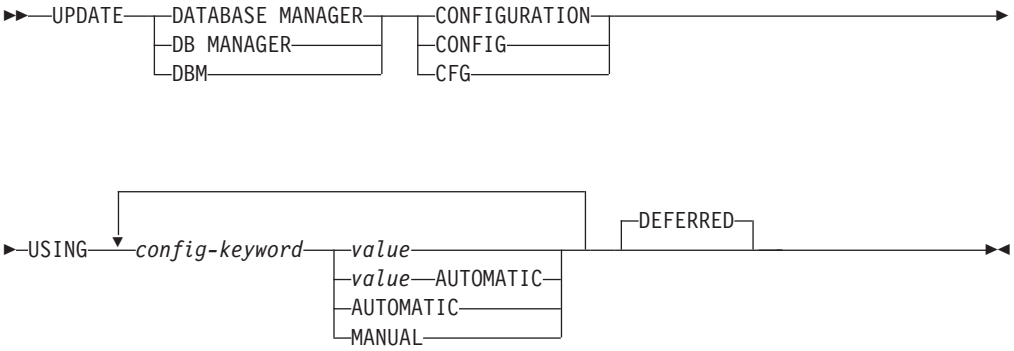
Authorization

SYSADM

Required connection

Database

Command syntax



Command parameters

AUTOMATIC
Some configuration parameters can be set to AUTOMATIC, allowing DB2 to automatically adjust these parameters to reflect the current resource requirements. For a list of configuration parameters that support the **AUTOMATIC** keyword, refer to the configuration parameters summary. If a value is specified along with the **AUTOMATIC** keyword, it might influence the automatic calculations. For specific details about this behavior, refer to the documentation for the configuration parameter.

Note: Note that the **federated_async** database manager configuration parameter can only be set to AUTOMATIC using the command line processor.

DEFERRED
Make the changes only in the configuration file, so that the changes take effect when the instance is restarted. This is the default.

This is a default clause when operating in the CLPPlus interface. **DEFERRED** need not be called when using CLPPlus processor.

MANUAL
Disables automatic tuning for the configuration parameter. The parameter is set to its current internal value and is no longer updated automatically.

USING *config-keyword value*
Specifies the database manager configuration parameter to be updated. For a list of configuration parameters, refer to the configuration parameters summary. *value* specifies the value to be assigned to the parameter.

Example

Update the diagnostic level to 1 for the database manager configuration.

```
CALL SYSPROC.ADMIN_CMD('db2 update dbm cfg using DIAGLEVEL 1')
```

Usage notes

To view or print a list of the database manager configuration parameters, use the SYSIBMADM.DBMCFG administrative view. To reset the database manager configuration parameters to the recommended database manager defaults, use the RESET DATABASE MANAGER CONFIGURATION command through the ADMIN_CMD procedure. For more information about database manager

configuration parameters and the values of these parameters appropriate for each type of database node configured (server, client, or server with remote clients), see individual configuration parameter descriptions.

Not all parameters can be updated.

Some changes to the database manager configuration file become effective only after they are loaded into memory. For more information on which parameters are configurable online and which ones are not, see the configuration parameter summary. Server configuration parameters that are not reset immediately are reset during execution of db2start. For a client configuration parameter, parameters are reset the next time you restart the application. If the client is the command line processor, it is necessary to invoke TERMINATE.

If an error occurs, the database manager configuration file does not change.

The database manager configuration file cannot be updated if the checksum is invalid. This can occur if you edit database manager configuration file and do not use the appropriate command. If the checksum is invalid, you must reinstall the database manager to reset the database manager configuration file.

When you update the **SVCENAME**, or **TPNAME** database manager configuration parameters for the current instance, if LDAP support is enabled and there is an LDAP server registered for this instance, the LDAP server is updated with the new value or values.

Command execution status is returned in the SQLCA resulting from the CALL statement.

Updates can only be made to the database instance that contains the connected database.

If a parameter supports dynamic update, an attempt is made to update it dynamically, even if the **IMMEDIATE** keyword is not specified. The authorization used is the current SYSTEM_USER id.

UPDATE HEALTH NOTIFICATION CONTACT LIST command using the ADMIN_CMD procedure

Updates the contact list for notification about health alerts issued by an instance.

Authorization

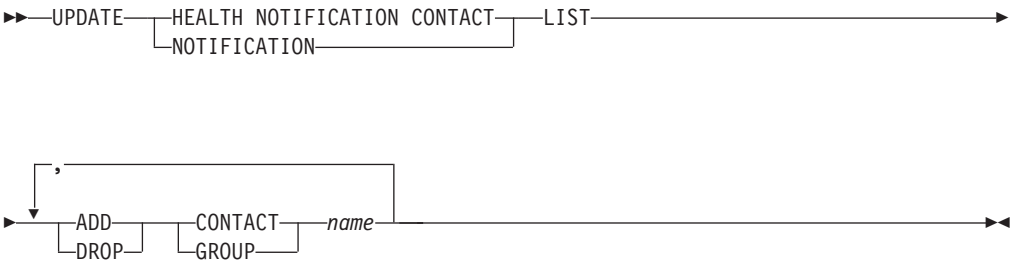
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT

Required Connection

Database

Command Syntax



Command Parameters

ADD GROUP *name*
Add a new contact group that will notified of the health of the instance.

ADD CONTACT *name*
Add a new contact that will notified of the health of the instance.

DROP GROUP *name*
Removes the contact group from the list of contacts that will notified of the health of the instance.

DROP CONTACT *name*
Removes the contact from the list of contacts that will notified of the health of the instance.

Example

Add the contact group 'gname1' to the health notification contact list:

```
CALL SYSPROC.ADMIN_CMD( 'update notification list add group gname1' )
```

Usage note

Command execution status is returned in the SQLCA resulting from the CALL statement.

UPDATE HISTORY command using the ADMIN_CMD procedure

Updates the location, device type, comment, or status in a database history records entry on the currently connected database partition.

Authorization

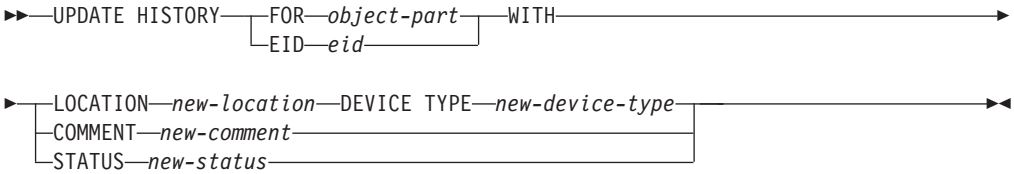
One of the following:

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM

Required connection

Database

Command syntax



Command parameters

FOR *object-part*
Specifies the identifier for the history entry to be updated. It is a time stamp with an optional sequence number from 001 to 999. This parameter cannot be used to update the entry status. To update the entry status, specify an EID instead.

EID *eid*
Specifies the history entry ID.

LOCATION *new-location*
Specifies the new physical location of a backup image. The interpretation of this parameter depends on the device type.

DEVICE TYPE *new-device-type*
Specifies a new device type for storing the backup image. Valid device types are:

D Disk

K Diskette

T Tape

A Tivoli Storage Manager

F Snapshot backup

U User exit

P Pipe

N Null device

X XBSA

Q SQL statement

O Other

COMMENT *new-comment*
Specifies a new comment to describe the entry.

STATUS *new-status*
Specifies a new status for an entry. Only backup entries can have their status updated. Valid values are:

A Active. The backup image is on the active log chain. Most entries are active.

I Inactive. Backup images that no longer correspond to the current log sequence, also called the current log chain, are flagged as inactive.

E Expired. Backup images that are no longer required, because there are more than NUM_DB_BACKUPS active images, are flagged as expired.

D Deleted. Backup images that are no longer available for recovery should be marked as having been deleted.

X Do not delete. Recovery database history records file entries that are marked DB2HISTORY_STATUS_DO_NOT_DELETE will not be pruned by calls to the PRUNE HISTORY command, running the ADMIN_CMD procedure with PRUNE HISTORY, calls to the db2Prune API, or automated recovery database history records pruning. You can use the DB2HISTORY_STATUS_DO_NOT_DELETE status to protect key recovery file entries from being pruned and the recovery objects associated with them from being deleted. Only log files, backup images, and load copy images can be marked as DB2HISTORY_STATUS_DO_NOT_DELETE.

Example

To update the database history records entry for a full database backup taken on April 13, 1997 at 10:00 a.m., enter:

```
CALL SYSPROC.ADMIN_CMD('update history
   for 19970413100000001 with location
   /backup/dbbackup.1 device type d')
```

Usage notes

The primary purpose of the database history records is to record information, but the data contained in the history is used directly by automatic restore operations. During any restore where the **AUTOMATIC** option is specified, the history of backup images and their locations will be referenced and used by the restore utility to fulfill the automatic restore request. If the automatic restore function is to be used and backup images have been relocated since they were created, it is recommended that the database history record for those images be updated to reflect the current location. If the backup image location in the database history is not updated, automatic restore will not be able to locate the backup images, but manual restore commands can still be used successfully.

Command execution status is returned in the SQLCA resulting from the CALL statement.

The *object-part* or *eid* must refer to the log history entries on the connected database partition.

UPDATE STMM TUNING DBPARTITIONNUM command using the ADMIN_CMD procedure

Update the user preferred self tuning memory manager (STMM) tuning database partition.

Authorization

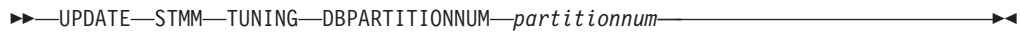
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- DBADM
- DATAACCESS
- SQLADM

Required connection

Database

Command syntax



Command parameter

partitionnum
partitionnum is an integer. If -1 or a nonexistent database partition number is used, DB2 will automatically select an appropriate database partition on which to run the STMM memory tuner.

Example

Update the user preferred self tuning memory manager (STMM) tuning database partition to database partition 3.

```
CALL SYSPROC.ADMIN_CMD( 'update stmm tuning dbpartitionnum 3' )
```

Usage notes

The STMM tuning process periodically checks for a change in the user preferred STMM tuning database partition number value. The STMM tuning process will move to the user preferred STMM tuning database partition if *partitionnum* exists and is an active database partition. Once this command changes the STMM tuning database partition number an immediate change is made to the current STMM tuning database partition number.

Command execution status is returned in the SQLCA resulting from the CALL statement.

This command commits its changes in the ADMIN_CMD procedure.

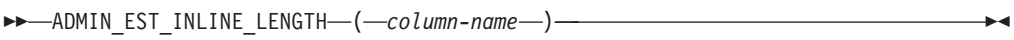
ADMIN_EST_INLINE_LENGTH function - Estimate length required to inline data

The ADMIN_EST_INLINE_LENGTH function returns an estimate of the inline length that is required to inline the data stored in an XML column, BLOB column, CLOB column, or DBCLOB column.

If the data cannot be inlined, the function returns a negative value.

If the data is already inlined, the function returns the actual length of the inlined data.

Syntax



The schema is SYSIBM.

Return value

This function returns either an INTEGER value that represents the estimated inline length (in bytes) of the data, or one of the following values:

NULL Indicates that the inputs are NULL.

-1 Indicates that the data cannot be inlined because there is no valid inline length that would allow the column value to be inlined.

-2 Indicates that the estimated inline length of the document cannot be determined because the document was inserted and stored in a release prior to DB2 for Linux, UNIX, and Windows Version 9.7.

Function parameters

column-name
Identifies a column of the base table with a data type of XML, BLOB, CLOB, or DBCLOB (SQLSTATE 42884). The column must directly or indirectly reference the column of a base table that is not generated based on an expression (SQLSTATE 42815).

Example

Example 1: The following example returns the estimated inline length of three XML documents that are contained in XML column xml_doc1 of TAB1 table.

```
db2 => SELECT PK, ADMIN_IS_INLINED(xml_doc1) as IS_INLINED,
              ADMIN_EST_INLINE_LENGTH(xml_doc1) as EST_INLINE_LENGTH
       from TAB1
```

This query results in the following output:

```
PK          IS_INLINED EST_INLINE_LENGTH
----------- ---------- -----------------
          1          1               292
          2          0               450
          3          0               454

  3 record(s) selected.
```

In the example, the ADMIN_IS_INLINED function indicates that the first document is inlined. Therefore, the ADMIN_EST_INLINE_LENGTH function returns the actual length of the inlined XML document. The second document is not inlined, so the ADMIN_EST_INLINE_LENGTH function returns the estimated inline length that is required to inline the second XML document.

Example 2: The following example returns the estimated inline length of one XML document that is contained in the XML column xml_doc1 of the TAB1 table. This example includes a predicate.

```
db2 => SELECT PK, ADMIN_IS_INLINED(xml_doc1) as IS_INLINED,
              ADMIN_EST_INLINE_LENGTH(xml_doc1) as EST_INLINE_LENGTH
       from TAB1 where PK=2
```

This query results in the following output:

```
PK          IS_INLINED EST_INLINE_LENGTH
----------- ---------- -----------------
          2          0               450

  1 record(s) selected.
```

Example 3: The following example returns the estimated inline length of three CLOB data that are contained in CLOB column clob_1 of the TAB1 table.

```
db2 => SELECT PK, ADMIN_IS_INLINED(clob_1) as IS_INLINED,
              ADMIN_EST_INLINE_LENGTH(clob_1) as EST_INLINE_LENGTH
       from TAB1
```

This query results in the following output:

```
PK          IS_INLINED EST_INLINE_LENGTH
----------- ---------- -----------------
          1          1                68
          2          0              3665
          3          0                -1

   3 record(s) selected.
```

Usage notes

- XML columns are only supported when the XML documents were inserted using DB2 for Linux, UNIX, and Windows Version 9.7 or later. XML documents inserted prior to this release have a different storage format. When the ADMIN_EST_INLINE_LENGTH function encounters an incorrect storage format, it returns a value of -2.
- If you plan to increase the column inline length, remember that this length cannot be reduced.
- Increasing the inline length also increases the total row size and might affect the performance of buffer pools. The total row size has the following limits.

Table 68. Row size limits

Page size	Row size limit	Inline length limit
4K	4005	4001
8K	8101	8097
16K	16 293	16 289
32K	32 677	32 673

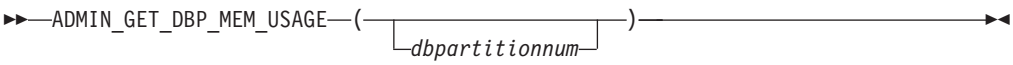
- The estimated inline length might not be accurate if the XML storage object page size is not same as the base table page size.

ADMIN_GET_DBP_MEM_USAGE table function - Get total memory consumption for instance

The ADMIN_GET_DBP_MEM_USAGE table function gets the total memory consumption for a given instance.

The *ADMIN_GET_DBP_MEM_USAGE* table function takes an optional input argument *dbpartitionnum* (INTEGER type), which specifies a valid database partition number, and returns only statistics for that single database partition. If the argument is omitted, statistics are returned for all active database partitions. When in a partitioned database environment, if you specify -1 or a NULL value for *dbpartitionnum*, data is returned from the currently connected partition.

Syntax



The schema is SYSPROC.

Table function parameters

dbpartitionnum
An optional input argument of type integer that specifies the database partition from which the memory usage statistics will be retrieved. If -1 or the NULL value is specified, data will be returned from the currently connected partition.

Authorization

EXECUTE privilege on the ADMIN_GET_DBP_MEM_USAGE function.

Information returned

Table 69. The result set for ADMIN_GET_DBP_MEM_USAGE

Column Name	Data Type	Description
DBPARTITIONNUM	SMALLINT	The database partition number from which memory usage statistics is retrieved.
MAX_PARTITION_MEM	BIGINT	The maximum amount of instance memory (in bytes) allowed to be consumed in the database partition if an instance memory limit is enforced.
CURRENT_PARTITION_MEM	BIGINT	The amount of instance memory (in bytes) currently consumed in the database partition.
PEAK_PARTITION_MEM	BIGINT	The peak or high watermark consumption of instance memory (in bytes) in the database partition.

Examples

Example 1: Retrieve memory usage statistics from database partition 3

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_DBP_MEM_USAGE(3)) AS T

DBPARTITIONNUM   MAX_PARTITION_MEM   CURRENT_PARTITION_MEM   PEAK_PARTITION_MEM
--------------   -----------------   ---------------------   ------------------
             3           500000000               381000000            481000000

  1 record(s) selected.
```

Example 2: Retrieve memory usage statistics from the currently connected partition (assuming the user is connected to the database at partition 2.)

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_DBP_MEM_USAGE(-1)) AS T

DBPARTITIONNUM   MAX_PARTITION_MEM   CURRENT_PARTITION_MEM   PEAK_PARTITION_MEM
--------------   -----------------   ---------------------   ------------------
             2           500000000               381000000            481000000

  1 record(s) selected.
```

Example 3: Retrieve memory usage statistics from all partitions

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_DBP_MEM_USAGE()) AS T

DBPARTITIONNUM   MAX_PARTITION_MEM   CURRENT_PARTITION_MEM   PEAK_PARTITION_MEM
--------------   -----------------   ---------------------   ------------------
             0           500000000               381000000            481000000
             1           500000000               381000000            481000000
             2           500000000               381000000            481000000
             3           500000000               381000000            481000000

  4 record(s) selected.
```

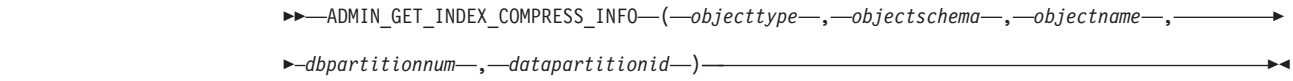
Example 4: Retrieve memory usage statistics in megabyte (MB) values

```
SELECT DBPARTITIONNUM, MAX_PARTITION_MEM/1048576 AS MAX_MEM_MB,
   CURRENT_PARTITION_MEM/1048576 AS CURRENT_MEM_MB, PEAK_PARTITION_MEM/1048576
   AS PEAK_MEM_MB FROM TABLE (SYSPROC.ADMIN_GET_DBP_MEM_USAGE()) AS T

DBPARTITIONNUM  MAX_MEM_MB  CURRENT_MEM_MB  PEAK_MEM_MB
--------------  ----------  --------------  -----------
             0        4590            1107         1107
             1        4590            1108         1108
             2        4590            1106         1106

  3 record(s) selected.
```

ADMIN_GET_INDEX_COMPRESS_INFO table function - returns compressed index information

The ADMIN_GET_INDEX_COMPRESS_INFO table function returns the potential index compression savings for uncompressed indexes or reports the index compression statistics from the catalogs.

Syntax



The schema is SYSPROC.

Table function parameters

objecttype
An input argument of type VARCHAR(1) that indicates the object type. The value must be one of the following case-sensitive values:

- 'T', NULL, or the empty string to indicate a table
- 'I' for an index

objectschema
A case-sensitive input parameter of type VARCHAR(128) that specifies the object schema.

If *objecttype* is 'T', NULL, or the empty string (''), then *objectschema* indicates the table schema.

- If *objectschema* is specified and *objectname* is NULL or the empty string (''), then information is returned for all indexes on all tables in the specified schema.
- If both *objectschema* and *objectname* are specified, then information is returned for all indexes on the specified table.

If *objecttype* is 'T', then *objectschema* indicates the index schema.

- If *objectschema* is specified and *objectname* is NULL or the empty string (''), then information is returned for all indexes in the specified schema.
- If both *objectschema* and *objectname* are specified, then information is returned for the specified index.
- If neither *objectschema* or *objectname* are specified, then information is returned for all indexes in all of the schemas.

If *objectname* is specified and *objectschema* is not specified, the function returns an SQL error. A parameter value is said to be unspecified when either it has a value of NULL or the empty string ('').

objectname
A case-sensitive input parameter of type VARCHAR(128) that specifies the object name. See the description for the *objectschema* parameter.

dbpartitionnum
An input parameter of type INTEGER that specifies a database partition number. When specified, information is returned only for indexes that reside on the specified database partition. To specify that data should be returned for all active database partitions, set the *dbpartitionnum* parameter value to either -2 or NULL. In nonpartitioned database environments, specify -2 or NULL.

datapartitionid
An input parameter of type INTEGER that specifies the data partition ID. When specified, information is returned only for index partitions defined on the specified data partitions. The data partition ID should correspond to the DATAPARTITIONID found in the SYSCAT.DATAPARTITIONS view. To specify that data should be returned for all data partitions, set the *datapartitionid* parameter value to either -2 or NULL. For nonpartitioned indexes, specify -2, 0, or NULL.

Authorization

EXECUTE privilege on the ADMIN_GET_INDEX_COMPRESS_INFO table function.

Example

After database migration, all the existing indexes are uncompressed. You may want to estimate the potential index compression savings for existing indexes on the table "S.T1", which has a data partition ID of 3 and resides on database partition number 2. In this example, S is the schema name and T1 is the table name, and T1 is not compressed

```
SELECT compress_attr, iid, dbpartitionnum, index_compressed,
    pct_pages_saved, num_leaf_pages_saved
  FROM TABLE(sysproc.admin_get_index_compress_info('', 'S', 'T1', 2, 3))
  AS t
```

The following is a sample of the output from this statement.

```
COMPRESS_ATTR       IID DBPARTITIONNUM INDEX_COMPRESSED ...
------------ --------- ------------- ---------------- ...
N                     1             2 N                ...
N                     2             2 N                ...

... PCT_PAGES_SAVED NUM_LEAF_PAGES_SAVED
... --------------- --------------------
...              50                  200
...              45                  150
```

You may decide that the savings from compression are worthwhile, and you want to enable index compression.

```
ALTER INDEX INDEX1 compress yes
ALTER INDEX INDEX2 compress yes
REORG INDEXES all FOR table S.T1
```

As time passes, you may determine the need to create new indexes for the table and want to estimate index compression savings for these indexes before compressing them. You may also wish to see the compression statistics from already compressed indexes.

```
SELECT compress_attr, iid, dbpartitionnum, index_compressed,
    pct_pages_saved,  num_leaf_pages_saved
  FROM TABLE(sysproc.admin_get_index_compress_info('', 'S', 'T1', 2, 3))
  AS t
```

The following is a sample of the output from this statement.

```
COMPRESS_ATTR       IID DBPARTITIONNUM INDEX_COMPRESSED ...
------------- --------- -------------- ---------------- ...
Y                     1              2 Y                ...
Y                     2              2 Y                ...
N                     3              2 N                ...
N                     4              2 N                ...

... PCT_PAGES_SAVED NUM_LEAF_PAGES_SAVED
... --------------- --------------------
...              -1                   -1
...              -1                   -1
...              58                  230
...              49                  140
```

As the first two indexes were already compressed, as indicated by the index_compressed column, the statement returns values from the system catalogs. In this case, the values from the catalogs were not collected.

After running RUNSTATS on the table, the next run of the index function yields the corrected results.

```
RUNSTATS ON TABLE S.T1 FOR INDEXES ALL
SELECT compress_attr, iid, dbpartitionnum, index_compressed,
    pct_pages_saved, num_leaf_pages_saved
  FROM TABLE(sysproc.admin_get_index_compress_info('', 'S', 'T1', 2, 3))
  AS t
```

The following is a sample of the output from this statement.

```
COMPRESS_ATTR       IID DBPARTITIONNUM INDEX_COMPRESSED ...
------------- --------- -------------- ---------------- ...
Y                     1              2 Y                ...
Y                     2              2 Y                ...
N                     3              2 N                ...
N                     4              2 N                ...

... PCT_PAGES_SAVED NUM_LEAF_PAGES_SAVED
... --------------- --------------------
...              50                  200
...              45                  150
...              58                  230
...              49                  140
```

ADMIN_GET_INDEX_COMPRESS_INFO table function metadata

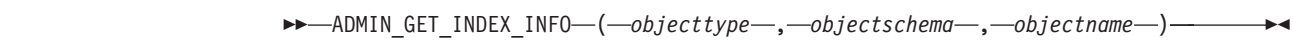
Table 70. ADMIN_GET_INDEX_COMPRESS_INFO table function metadata

Column Name	Data Type	Description
INDSCHEMA	VARCHAR(128)	Name of the schema where the index is defined.
INDNAME	VARCHAR(128)	Index name.
TABSCHEMA	VARCHAR(128)	Name of the schema where the table is defined.
TABNAME	VARCHAR(128)	Table name.
DBPARTITIONNUM	SMALLINT	Database partition number.
IID	SMALLINT	Identifier for the index.
DATAPARTITIONID	INTEGER	Data partition ID.
COMPRESS_ATTR	CHAR(1)	The state of the COMPRESSION attribute on the index. • "Y" = Index compression is enabled • "N" = Index compression is not enabled
INDEX_COMPRESSED	CHAR(1)	Physical index format. • "Y" = Index is in compressed format • "N" = Index is in uncompressed format If the physical index format does not match the compression attribute, an index reorganization is needed to convert index to the defined format If the table or index is in error at the time this function is executed, then this value is NULL.
PCT_PAGES_SAVED	SMALLINT	If the index is not physically compressed (INDEX_COMPRESSED is "N"), then this value represents the estimated percentage of leaf pages saved, as if the index were actually compressed. If the index is physically compressed (INDEX_COMPRESSED is "Y"), then this value reports the PCTPAGESSAVED value from the system catalog view (either SYSCAT.INDEXES or SYSCAT.INDEXPARTITIONS). **Note:** This value is the same for each entry of an index or index partition for each database partition in a partitioned database environment. If the table or index is in error at the time this function is executed, then this value is NULL.
NUM_LEAF_PAGES_SAVED	BIGINT	If the index is not physically compressed (INDEX_COMPRESSED is "N"), then this value represents the estimated number of leaf pages saved as if the index were actually compressed. If the index is physically compressed (INDEX_COMPRESSED is "Y"), then this value reports the calculated number of leaf pages saved, based on the PCTPAGESSAVED and NLEAF values from the system catalog view (either SYSCAT.INDEXES or SYSCAT.INDEXPARTITIONS). If either PCTPAGESSAVED or NLEAF are invalid values (-1), then this value is set to -1 as well. **Note:** This value is the same for each entry of an index or index partition for each database partition in a partitioned database environment. If the table or index is in error at the time this function is executed, then this value is NULL.

ADMIN_GET_INDEX_INFO table function - returns index information

The ADMIN_GET_INDEX_INFO table function returns index information not available in the catalog views, such as compression information and the logical and physical size of the index.

Syntax



The schema is SYSPROC.

Table function parameters

objecttype
An input argument of type VARCHAR(1) that indicates the object type. The value must be one of the following case-sensitive values:

- 'T', NULL, or the empty string ('') to indicate a table
- 'I' for an index

objectschema
A case-sensitive input parameter of type VARCHAR(128) that specifies the object schema.

If *objecttype* is 'T', NULL, or the empty string (''), then*objectschema* indicates the table schema.

- If *objectschema* is specified and *objectname* is NULL or the empty string (''), then information is returned for all indexes on all tables in the specified schema.
- If both *objectschema* and *objectname* are specified, then information is returned for all indexes on the specified table.

If *objecttype* is 'I', then *objectschema* indicates the index schema.

- If *objectschema* is specified and *objectname* is NULL or the empty string (''), then information is returned for all indexes in the specified schema.
- If both *objectschema* and *objectname* are specified, then information is returned for the specified index.
- If neither *objectschema* or *objectname* are specified, then information is returned for all indexes in all of the schemas.

If *objectname* is specified and *objectschema* is not specified, the function returns an SQL error. A parameter value is said to be unspecified when either it has a value of NULL or the empty string ('').

objectname
A case-sensitive input parameter of type VARCHAR(128) that specifies the object name. See the description for the *objectschema* parameter.

Authorization

EXECUTE privilege on the ADMIN_GET_INDEX_INFO table function.

Example

After enabling index compression for several indexes on a table, you want to determine which indexes are compressed and which indexes require a rebuild in order to be compressed. In this example, S is the schema name and T1 is the table name.

```
db2 SELECT iid, compress_attr, index_compressed
    FROM TABLE(sysproc.admin_get_index_info('','S','T1')) AS t
```

The following is an example of output from this query.

```
IID       COMPRESS_ATTR INDEX_COMPRESSED
--------- ------------- ----------------
        1 Y             Y
        2 Y             Y
        3 Y             N
        4 N             N
```

Additionally, you want to see other index information for all indexes in the schema S2. In this example:

- T2 = a partitioned table with two data partitions
- T3 = a nonpartitioned table
- IND_1 = a nonpartitioned index on T2
- IND_2 = a partitioned index on T2
- IND_3 = a partitioned index on T2
- IND_4 = an index on T3
- IND_5 = an index on T3

```
db2 SELECT tabname, indname, iid,index_partitioning, datapartitionid,
   index_object_l_size, index_object_p_size, index_requires_rebuild,
   large_rids, FROM TABLE(sysproc.admin_get_index_info('I','S2','')) AS t
```

The following is an example of the output from this query.

```
TABNAME INDNAME IID       INDEX_PARTITIONING DATAPARTITIONID
------- ------- --------- ------------------ ---------------
T2      IND_1           1 N                              0
T2      IND_2           2 P                              1
T2      IND_2           2 P                              2
T2      IND_3           3 P                              1
T2      IND_3           3 P                              2
T3      IND_4           4                                0
T3      IND_5           5                                0
```

Output from this procedure (continued):

```
INDEX_OBJECT_L_SIZE INDEX_OBJECT_P_SIZE INDEX_REQUIRES_REBUILD LARGE_RIDS
------------------- ------------------- ---------------------- ----------
                 50                  51                      N          Y
                 40                  40                      N          Y
                 45                  45                      N          Y
                 40                  40                      N          Y
                 45                  45                      N          Y
                 20                  20                      N          Y
                 20                  20                      N          Y
```

ADMIN_GET_INDEX_INFO table function metadata

Table 71. ADMIN_GET_INDEX_INFO table function metadata

Column Name	Data Type	Description
INDSCHEMA	VARCHAR(128)	Name of the schema where the index is defined.
INDNAME	VARCHAR(128)	Index name.
TABSCHEMA	VARCHAR(128)	Name of the schema where the table is defined.
TABNAME	VARCHAR(128)	Table name.
DBPARTITIONNUM	SMALLINT	Database partition number.
IID	SMALLINT	Identifier for the index.
DATAPARTITIONID	INTEGER	Data partition ID.

Table 71. ADMIN_GET_INDEX_INFO table function metadata (continued)

Column Name	Data Type	Description
COMPRESS_ATTR	CHAR(1)	The state of the COMPRESSION attribute on the index. • "Y" = Index compression is enabled • "N" = Index compression is not enabled
INDEX_COMPRESSED	CHAR(1)	Physical index format. • "Y" = Index is in compressed format • "N" = Index is in uncompressed format If the physical index format does not match the compression attribute, an index reorganization is needed to convert the index to the defined format. If the table or index is in error when this function is executed, then this value is NULL.
INDEX_PARTITIONING	CHAR(1)	Identifies the partitioning characteristic of the index. • "N" = Nonpartitioned index • "P" = Partitioned index • Blank = Index is not on a partitioned table
INDEX_OBJECT_L_SIZE	BIGINT	Logical size of the index object. For nonpartitioned tables, this is the amount of disk space logically allocated for all indexes defined on the table. For a nonpartitioned index on a partitioned table, this is the amount of disk space logically allocated for the index. For a partitioned index on a partitioned table, this is the amount of disk space logically allocated for all index partitions defined on the data partition. All sizes are reported in kilobytes (KB). The logical size is the amount of space that the table or data partition knows about. It may be less than the amount of space physically allocated to hold index data for the table or data partition (for example, in the case of a logical table truncation). The size returned takes into account full extents that are logically allocated for the indexes and, for indexes created in DMS table spaces, an estimate of the EMP extents. If the table or index is in error when this function is executed, then this value is NULL.
INDEX_OBJECT_P_SIZE	BIGINT	Physical size of the index object. For nonpartitioned tables, this is the amount of disk space physically allocated for all indexes defined on the table. For a nonpartitioned index on a partitioned table, this is the amount of disk space physically allocated for the index. For a partitioned index on a partitioned table, this is the amount of disk space physically allocated for all index partitions defined on the data partition. All sizes are reported in kilobytes (KB). The size returned takes into account full extents allocated for the indexes and includes the EMP extents for indexes created in DMS table spaces. If the table or index is in error when this function is executed, then this value is NULL.
INDEX_REQUIRES_REBUILD	CHAR(1)	Rebuild status for the index. • "Y" if the index defined on the table or data partition requires a rebuild • "N" otherwise If the table is in error when this function is executed, then this value is NULL.

Table 71. ADMIN_GET_INDEX_INFO table function metadata (continued)

Column Name	Data Type	Description
LARGE_RIDS	CHAR(1)	Indicates whether or not the index is using large row IDs (RIDs) (4 byte page number, 2 byte slot number). • "Y" indicates that the index is using large RIDs • "N" indicates that the index is not using large RIDs • "P" (pending) indicates that the table that the index is defined on supports large RIDs (that is, the table is in a large table space), but the index for the table or data partition has not been reorganized or rebuilt yet. Therefore, the table is still using 4 byte RIDs, and action must be taken to convert the table or index to large RIDs. If the table is in error where this function is executed, then this value is NULL.

ADMIN_GET_MSGS table function - Retrieve messages generated by a data movement utility that is executed through the ADMIN_CMD procedure

The ADMIN_GET_MSGS table function is used to retrieve messages generated by a single execution of a data movement utility command through the ADMIN_CMD procedure. The input parameter *operation_id* identifies that operation.

Syntax

The schema is SYSPROC.

Table function parameter

operation_id
An input argument of type VARCHAR(139) that specifies the operation ID of the message file(s) produced by a data movement utility that was executed through the ADMIN_CMD procedure. The operation ID is generated by the ADMIN_CMD procedure.

Authorization

EXECUTE privilege on the ADMIN_GET_MSGS table function. The fenced user ID must have read access to the files under the directory indicated by registry variable DB2_UTIL_MSGPATH. If the registry variable is not set, then the fenced user ID must have read access to the files in the `tmp` subdirectory of the instance directory.

Example

Check all the messages returned by EXPORT utility that was executed through ADMIN_CMD procedure, with operation ID '24523_THERESAX'

```
SELECT * FROM TABLE(SYSPROC.ADMIN_GET_MSGS('24523_THERESAX')) AS MSG
```

The following is an example of output from this query.

```
DBPARTITIONNUM AGENTTYPE SQLCODE   MSG
-------------- --------- --------- ---------------------------------------...-
-              -         SQL3104N  The Export utility is beginning to
                                    export data to file
                                    "/home/theresax/rtest/data/ac_load03.del".
-              -         SQL3105N  The Export utility has finished
                                    exporting "8" rows.

2 record(s) selected.
```

Usage notes

The query statement that invokes this table function with the appropriate *operation_id* can be found in the MSG_RETRIEVAL column of the first result set returned by the ADMIN_CMD procedure.

Information returned

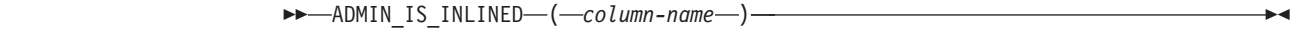
Table 72. Information returned by the ADMIN_GET_MSGS table function

Column name	Data type	Description
DBPARTITIONNUM	INTEGER	Database partition number. This value is only returned for a distributed load and indicates which database partition the corresponding message is for.
AGENTTYPE	CHAR(4)	Agent type. This value is only returned for a distributed load. The possible values are: • 'LOAD': for load agent • 'PART': for partitioning agent • 'PREP': for pre-partitioning agent • NULL: no agent type information is available
SQLCODE	VARCHAR(9)	SQLCODE of the message being returned.
MSG	VARCHAR(1024)	Short error message that corresponds to the SQLCODE.

ADMIN_IS_INLINED function - Determine if data is inlined

The ADMIN_IS_INLINED function retrieves state information about inline data for an XML column, BLOB column, CLOB column, or DBCLOB column.

Syntax



The schema is SYSIBM.

Return value

This function returns one of the following values of type SMALLINT, or the null value:

1 Indicates that the data is inlined.

0 Indicates that the data is not inlined.

NULL Indicates that the inputs are NULL.

Function parameters

column-name
Identifies a column of the base table with a data type of XML, BLOB, CLOB, or DBCLOB (SQLSTATE 42884). The column must directly or indirectly reference the column of a base table that is not generated based on an expression (SQLSTATE 42815).

Example

Example 1: The following example indicates whether the three XML documents in the XML column xml_doc1 of the TAB1 table are inlined:

```
db2 => SELECT PK, ADMIN_IS_INLINED(xml_doc1) as IS_INLINED
       from TAB1
```

This query results in the following output:

```
PK          IS_INLINED
----------- -------------------
          1                   1
          2                   0
          3                   0

  3 record(s) selected.
```

Example 2: The following example indicates whether one of the XML documents in the XML column xml_doc1 of the TAB1 table is inlined:

```
db2 => SELECT PK, ADMIN_IS_INLINED(xml_doc1) as IS_INLINED
       from TAB1 where PK=1
```

This query results in the following output:

```
PK          IS_INLINED
----------- -------------------
          1                   1

  1 record(s) selected.
```

Example 3: The following example indicates whether the three CLOB data contained in the CLOB column clob_1 of the TAB1 table are inlined:

```
db2 => SELECT PK, ADMIN_IS_INLINED(clob_1) as IS_INLINED
       from TAB1
```

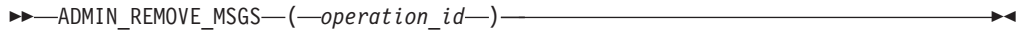
This query results in the following output:

```
PK          IS_INLINED
----------- -------------------
          1                   0
          2                   0
          3                   1

  3 record(s) selected.
```

ADMIN_REMOVE_MSGS procedure - Clean up messages generated by a data movement utility that is executed through the ADMIN_CMD procedure

The ADMIN_REMOVE_MSGS procedure is used to clean up messages generated by a single execution of a data movement utility command through the ADMIN_CMD procedure. The input parameter *operation_id* identifies the operation.

Syntax



The schema is SYSPROC.

Procedure parameter

operation_id
An input argument of type VARCHAR(139) that specifies the operation ID of the message file(s) produced by a data movement utility that was executed through the ADMIN_CMD procedure. The operation ID is generated by the ADMIN_CMD procedure.

Authorization

EXECUTE privilege on the ADMIN_REMOVE_MSGS procedure. The fenced user ID must be able to delete files under the directory indicated by registry variable DB2_UTIL_MSGPATH. If the registry variable is not set, then the fenced user ID must be able to delete the files in the `tmp` subdirectory of the instance directory.

Example

Clean up messages with operation ID '24523_THERESAX'.

```
CALL SYSPROC.ADMIN_REMOVE_MSGS('24523_THERESAX')
```

Usage notes

The CALL statement that invokes this procedure with the appropriate *operation_id* can be found in the MSG_REMOVAL column of the first result set returned by ADMIN_CMD procedure.

ADMIN_REVALIDATE_DB_OBJECTS procedure - Revalidate invalid database objects

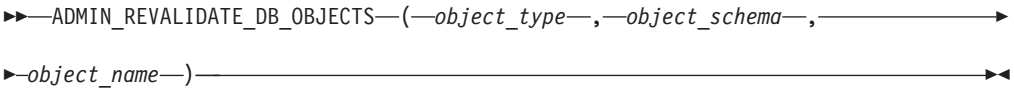
The ADMIN_REVALIDATE_DB_OBJECTS procedure revalidates invalid database objects.

This procedure takes three input parameters, *object_type*, *object_schema*, and *object_name*, that control the level of revalidation that is performed:

- To revalidate all of the invalid objects in the database, either specify NULL for all parameters, or call the procedure without parameters.
- To revalidate all of the invalid database objects under a specific schema, specify a value for *object_schema*, and specify NULL for *object_name* and *object_type*.

- To revalidate a specific invalid database object, specify valid values for all parameters.

Syntax



The schema is SYSPROC.

Procedure parameters

object_type
An input argument of type VARCHAR(30) that identifies the type of the database object. The following types are valid:

- FUNCTION
- GLOBAL VARIABLE
- METHOD
- MODULE
- PROCEDURE
- SPECIFIC
- TABLE
- TRIGGER
- TYPE
- VIEW

This value is not case-sensitive. This value can be NULL.

If any of these types is specified, the procedure revalidates all of the invalid objects of that type, with the exception of those that belong to a MODULE. If you want to revalidate objects that are inside of a module, use the MODULE type with the name of a specific module, and all of the invalid objects inside of that module will be revalidated.

If there is a routine that has more than one parameter signature and you only want to revalidate one of them, use the SPECIFIC type with the name of the routine that you want to revalidate.

If you use the TABLE type, the specified tables will be reorganized and their statistics will be collected. The procedure invokes the reorg utility, followed by the runstats utility, against regular or materialized query tables that are in reorg-pending state. The procedure will attempt to use a user profile for runstats, if one exists. If not, a default runstats operation is invoked.

object_schema
An input argument of type VARCHAR(128) that identifies the schema name used to qualify database object references. The name is case-sensitive. This value can be NULL.

object_name
An input argument of type VARCHAR(128) that identifies a database object. The name is case-sensitive. This value cannot be the value of a typed table or a row function, because the procedure does not support these types of objects; if the name of such an object is specified, an error is returned. This value can be NULL.

Authorization

EXECUTE privilege on the ADMIN_REVALIDATE_DB_OBJECTS procedure.

Examples

Example 1: Revalidate everything in the current database.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS(NULL, NULL, NULL)
```

Or, alternatively, call the procedure without any parameters.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS()
```

Example 2: Revalidate all objects that are qualified by the schema MY_SCHEMA.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS(NULL, 'MY_SCHEMA', NULL)
```

Example 3: Revalidate all trigger objects in the database.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS('trigger', NULL, NULL)
```

Example 4: Revalidate a specific view object.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS('view', 'MY_SCHEMA', 'MY_VIEW')
```

Example 5: Revalidate all procedures under MY_SCHEMA. In this example, there are three procedures (proc1, proc2, and proc3) under this schema. The referenced object used by proc1 does not exist. The following call revalidates proc2 and proc3, but proc1 remains invalid. In this situation, the call returns a warning.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS('procedure', 'MY_SCHEMA', NULL)
```

Example 6: Revalidate an object that does not exist. This example returns an error.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS('procedure', 'MY_SCHEMA', 'MY_VIEW')
```

Example 7: Revalidate all procedures under MY_SCHEMA using the named parameter notation.

```
CALL SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS(
  object_type=>'PROCEDURE',object_schema=>'MY_SCHEMA')
```

Usage notes

All of the non-null parameter values that are passed to the ADMIN_REVALIDATE_DB_OBJECTS procedure must be satisfied, or the procedure cannot identify the objects that need to be revalidated. For example, if you specify a view name, but with a trigger type, the procedure does not revalidate the view, because the type does not match.

This procedure will revalidate only invalid objects and regular or materialized query tables in reorg-pending state. All invalid objects can be found in SYSCAT.INVALIDOBJECTS. To find out which tables are in reorg-pending state, use the ADMIN_GET_TAB_INFO table function.

If a valid object is specified as input, the procedure will not perform any operation and returns a success code. If a failure occurs during the revalidation of tables, the procedure fails. If a failure occurs during the revalidation of other objects, the procedure ignores the failure and continues revalidating the other objects. If there is at least one failure, the procedure returns a warning (SQLSTATE 0168B). If the

revalidation of all objects fails, the procedure returns an error (SQLSTATE 429C4). The details of all revalidation failures of objects except tables can be found in SYSCAT.INVALIDOBJECTS.

When a global variable is revalidated, it is also instantiated for the current session.

To monitor the progress of a table revalidation, you can monitor the progress of the associated table reorg operation. For all other objects, query the SYSCAT.INVALIDOBJECTS catalog view; objects are deleted from this view when they are successfully revalidated, and entries are updated if revalidation fails.

ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function - returns compressed information

The ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO_V97 table function return compression information for tables, materialized query tables (MQT) and hierarchy tables.

ADMINTABCOMPRESSINFO administrative view

The ADMINTABCOMPRESSINFO administrative view returns compression information for tables, materialized query tables (MQT) and hierarchy tables only. These table types are reported as T for table, S for materialized query tables and H for hierarchy tables in the SYSCAT.TABLES catalog view. The information is returned at both the data partition level and the database partition level for a table.

The schema is SYSIBMADM.

Refer to the \ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function metadata table for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the ADMINTABCOMPRESSINFO administrative view
- CONTROL privilege on the ADMINTABCOMPRESSINFO administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the ADMIN_GET_TAB_COMPRESS_INFO_V97 table function
- DATAACCESS authority

Examples

Example 1: Retrieve all compression information for all tables

```
SELECT * FROM SYSIBMADM.ADMINTABCOMPRESSINFO
```

The following is an example of output from this query:

```
TABSCHEMA TABNAME   DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER ...
--------- --------- -------------- ----------------- ------------- ------------ ...
SYSIBM    SYSTABLES              0                 0 N             NOT BUILT    ...
SYSIBM    SYSTABLES              0                 0 N             NOT BUILT    ...
...
SIMAP2    STAFF                  0                 4 Y             REORG        ...
SIMAP2    STAFF                  0                 4 Y             REORG        ...
...
  156 record(s) selected.
```

Output from this query (continued):

```
DICT_BUILD_TIMESTAMP       COMPRESS_DICT_SIZE EXPAND_DICT_SIZE ROWS_SAMPLED ...
-------------------------- ------------------ ---------------- ------------ ...
-                                           0                0            0 ...
-                                           0                0            0 ...
...
2009-03-31-11.08.18.000000               3968             3000            6 ...
2009-03-31-11.08.18.000000              13312            10944            6 ...
...
```

Output from this query (continued):

```
PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
------------------- ------------------- ------------------------ -----------
                  0                   0                        0 DATA
                  0                   0                        0 XML
...
                 70                  70                       31 DATA
                 66                  66                      235 XML
...
```

Example 2: Determine the object on which the dictionary was created, the dictionary building action, and the time of dictionary creation for all tables.

```
SELECT TABSCHEMA, TABNAME, DBPARTITIONNUM, DATA_PARTITION_ID,
    OBJECT_TYPE, DICT_BUILDER, DICT_BUILD_TIMESTAMP
  FROM SYSIBMADM.ADMINTABCOMPRESSINFO
```

The following is an example of output from this query:

```
TABSCHEMA       TABNAME         DBPARTITIONNUM DATA_PARTITION_ID ...
--------------- --------------- -------------- ----------------- ...
SYSIBM          SYSTABLES                    0                 0 ...
SYSIBM          SYSTABLES                    0                 0 ...
...
SIMAP2          STAFF                        0                 4 ...
SIMAP2          STAFF                        0                 4 ...
SYSTOOLS        HMON_COLLECTION              0                 0 ...
SYSTOOLS        HMON_COLLECTION              0                 0 ...

  156 record(s) selected.
```

Output from this query (continued):

```
OBJECT_TYPE DICT_BUILDER               DICT_BUILD_TIMESTAMP
----------- -------------------------- --------------------------
DATA        NOT BUILT                  -
XML         NOT BUILT                  -
...
DATA        REORG                      2009-03-31-11.08.18.000000
XML         REORG                      2009-03-31-11.08.18.000000
DATA        REDISTRIBUTE               2009-03-29-06.44.32.000000
XML         REDISTRIBUTE               2009-03-29-06.44.32.000000
```

ADMIN_GET_TAB_COMPRESS_INFO_V97 table function

The ADMIN_GET_TAB_COMPRESS_INFO_V97 table function returns the same information as the ADMINTABCOMPRESSINFO administrative view, but allows you to specify a schema, table name and an execution mode.

Refer to the ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function metadata table for a complete list of information that can be returned.

This function returns two rows for every table. One row has a value of DATA in the OBJECT_TYPE column, and the other row has a value of XML for that column. The row marked as DATA is equivalent to the return value from the deprecated "ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO" on page 1029 table function. The row marked as XML describes the XML compression dictionary.

Syntax

The schema is SYSPROC.

Table function parameters

tabschema
An input argument of type VARCHAR(128) that specifies a schema name.

tabname
An input argument of type VARCHAR(128) that specifies a table name, a materialized query table name or a hierarchy table name.

execmode
An input argument of type VARCHAR(30) that specifies the execution mode. The execution mode can be one of the following:

- 'REPORT' -- Reports compression information as of last generation. This is the default value.
- 'ESTIMATE' -- Generates new compression information based on the current table.

Authorization

EXECUTE privilege on the ADMIN_GET_TAB_COMPRESS_INFO_V97 function.

Examples

Example 1: Retrieve existing compression information for table SIMAP2.STAFF

```
SELECT *
  FROM TABLE (
    SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO_V97('SIMAP2', 'STAFF', 'REPORT'))
  AS T
```

The following is an example from output of this query:

```
TABSCHEMA TABNAME DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER ...
--------- ------- -------------- ----------------- ------------- ------------ ...
SIMAP2    STAFF                0                 4 Y             REORG        ...
SIMAP2    STAFF                0                 4 Y             NOT BUILT    ...

2 record(s) selected.
```

Output from this query (continued):

```
DICT_BUILD_TIMESTAMP       COMPRESS_DICT_SIZE EXPAND_DICT_SIZE ROWS_SAMPLED ...
-------------------------- ------------------ ---------------- ------------ ...
2009-03-31-12.19.30.000000              13312             5296           35 ...
-                                           0                0            0 ...
```

Output from this query (continued):

```
PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
------------------- ------------------- ------------------------ -----------
                 38                  38                       27 DATA
                  0                   0                        0 XML
```

Example 2: Retrieve estimated compression information for table SIMAP2.STAFF as of now.

```
SELECT *
  FROM TABLE (
    SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO_V97('SIMAP2', 'STAFF', 'ESTIMATE'))
  AS T
```

The following is an example from output of this query:

```
TABSCHEMA TABNAME DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER   ...
--------- ------- -------------- ----------------- ------------- -------------- ...
SIMAP2    STAFF                0                 4 Y             TABLE FUNCTION ...
SIMAP2    STAFF                0                 4 Y             TABLE FUNCTION ...

2 record(s) selected.
```

Output from this query (continued):

```
DICT_BUILD_TIMESTAMP       COMPRESS_DICT_SIZE EXPAND_DICT_SIZE ROWS_SAMPLED ...
-------------------------- ------------------ ---------------- ------------ ...
2009-03-31-12.27.06.000000              13312             5296           35 ...
2009-03-31-12.27.06.000000              13312             9544            8 ...
```

Output from this query (continued):

```
PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
------------------- ------------------- ------------------------ -----------
                 38                  38                       27 DATA
                 75                  75                       95 XML
```

Example 3: Determine the total dictionary size for all objects in tables in the schema SIMAP2

```
SELECT TABSCHEMA, TABNAME, OBJECT_TYPE, DICT_BUILDER, (
     COMPRESS_DICT_SIZE+EXPAND_DICT_SIZE)
     AS TOTAL_DICT_SIZE, DBPARTITIONNUM, DATA_PARTITION_ID
FROM TABLE (
  SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO_V97('SIMAP2', '', 'REPORT'))
AS T
```

Output from this query:

```
TABSCHEMA    TABNAME      OBJECT_TYPE  DICT_BUILDER ...
------------ ---------- -------------- ------------ ...
SIMAP2       ACT          DATA         NOT BUILT    ...
SIMAP2       ACT          XML          NOT BUILT    ...
```

```
SIMAP2       ADEFUSR      DATA         INSPECT      ...
SIMAP2       ADEFUSR      XML          NOT BUILT    ...
...
SIMAP2       CUSTOMER     DATA         REORG        ...
SIMAP2       CUSTOMER     XML          REORG        ...
SIMAP2       DEPARTMENT   DATA         NOT BUILT    ...
SIMAP2       DEPARTMENT   XML          NOT BUILT    ...
...
SIMAP2       STAFF        DATA         REORG        ...
SIMAP2       STAFF        XML          NOT BUILT    ...
SIMAP2       SUPPLIERS    DATA         TABLE GROWTH ...
SIMAP2       SUPPLIERS    XML          NOT BUILT    ...

44 record(s) selected.
```

Output from this query (continued):

```
TOTAL_DICT_SIZE DBPARTITIONNUM DATA_PARTITION_ID
--------------- -------------- -----------------
              0              0                 0
              0              0                 0
           1890              0                 0
              0              0                 0
...
           6968              0                 1
          24256              0                 1
              0              1                 0
              0              1                 0
...
          18608              0                 4
              0              0                 4
           6960              0                 2
              0              0                 2
```

Example 4: View a report of the dictionary information of tables in the SIMAP2 schema.

```
  SELECT * FROM TABLE (
      SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO_V97('SIMAP2', '', 'REPORT'))
    AS T
```

Output from this query:

```
TABSCHEMA TABNAME   DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER ...
--------- --------- -------------- ----------------- ------------- ------------ ...
SIMAP2    ACT                    0                 0 N             NOT BUILT    ...
SIMAP2    ACT                    0                 0 N             NOT BUILT    ...
SIMAP2    ADEFUSR                0                 0 N             INSPECT      ...
SIMAP2    ADEFUSR                0                 0 N             NOT BUILT    ...
...
SIMAP2    CUSTOMER               0                 1 Y             REORG        ...
SIMAP2    CUSTOMER               0                 1 Y             REORG        ...
...
SIMAP2    STAFF                  0                 4 Y             REORG        ...
SIMAP2    STAFF                  0                 4 Y             NOT BUILT    ...
SIMAP2    SUPPLIERS              0                 2 N             NOT BUILT    ...
SIMAP2    SUPPLIERS              0                 2 N             NOT BUILT    ...

44 record(s) selected.
```

Output from this query (continued):

```
DICT_BUILD_TIMESTAMP       COMPRESS_DICT_SIZE EXPAND_DICT_SIZE ROWS_SAMPLED ...
-------------------------- ------------------ ---------------- ------------ ...
-                                           0                0            0 ...
-                                           0                0            0 ...
2009-03-31-12.11.02.000000                290             1890           22 ...
-                                           0                0            0 ...
```

```
...
2009-03-31-11.08.18.000000              3968            3000           6 ...
2009-03-31-11.08.18.000000             13312           10944           6 ...
...
2009-03-31-12.19.30.000000             13312            5296          35 ...
-                                          0               0           0 ...
-                                          0               0           0 ...
-                                          0               0           0 ...
```

Output from this query (continued):

```
PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
------------------- ------------------- ---------------------- -----------
                  0                   0                      0 DATA
                  0                   0                      0 XML
                 20                  25                     21 DATA
                  0                   0                      0 XML
...
                 70                  70                     31 DATA
                 66                  66                    235 XML
...
                 38                  38                     27 DATA
                  0                   0                      0 XML
                  0                   0                      0 DATA
                  0                   0                      0 XML
```

Example 5: View a report of the dictionary information of DATA objects of tables in the SIMAP2 schema.

```
SELECT * FROM TABLE (
    SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO_V97('SIMAP2','','REPORT'))
  WHERE OBJECT_TYPE='DATA'
```

Output from this query:

```
TABSCHEMA TABNAME   DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER ...
--------- --------- -------------- ----------------- ------------- ------------ ...
SIMAP2    ACT                    0                 0 N             NOT BUILT    ...
SIMAP2    ADEFUSR                0                 0 N             INSPECT      ...
...
SIMAP2    CUSTOMER               0                 1 Y             REORG        ...
SIMAP2    DEPARTMENT             1                 0 N             NOT BUILT    ...
...
SIMAP2    STAFF                  0                 4 Y             REORG        ...
SIMAP2    SUPPLIERS              0                 2 N             NOT BUILT    ...

  22 record(s) selected.
```

Output from this query (continued):

```
DICT_BUILD_TIMESTAMP       COMPRESS_DICT_SIZE EXPAND_DICT_SIZE ROWS_SAMPLED...
-------------------------- ------------------ ---------------- ------------ ...
-                                           0                0            0 ...
2009-03-31-12.11.02.000000                290             1890           22 ...
...
2009-03-31-11.08.18.000000               3968             3000            6 ...
-                                           0                0            0 ...
...
2009-03-31-12.19.30.000000              13312             5296           35 ...
-                                           0                0            0 ...
```

Output from this query (continued):

```
PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
------------------- ------------------- ---------------------- -----------
                  0                   0                      0 DATA
                 20                  25                     21 DATA
```

```
                70                 70                    31 DATA
                 0                  0                     0 DATA

                38                 38                    27 DATA
                 0                  0                     0 DATA
```

Example 6: View a report of the dictionary information of XML objects of the CUSTOMER table in the SIMAP2 schema.

```
SELECT * FROM TABLE (
    SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO_V97('SIMAP2', 'CUSTOMER',  'REPORT'))
  WHERE OBJECT_TYPE='XML'
```

Output from this query:

```
TABSCHEMA TABNAME   DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER ...
--------- --------- -------------- ----------------- ------------- ------------ ...
SIMAP2    CUSTOMER               0                 1 Y             REORG        ...
```

Output from this query (continued):

```
DICT_BUILD_TIMESTAMP       COMPRESS_DICT_SIZE EXPAND_DICT_SIZE ROWS_SAMPLED ...
-------------------------- ------------------ ---------------- ------------ ...
2009-03-31-11.08.18.000000              13312            10944            6 ...
```

Output from this query (continued):

```
PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH OBJECT_TYPE
------------------- ------------------- ----------------------- -----------
                 66                  66                     235 XML
```

Usage notes

- If both the *tabschema* and *tabname* are specified, information is returned for that specific table only.
- If the *tabschema* is specified but *tabname* is empty (') or NULL, information is returned for all tables in the given schema.
- If the *tabschema* is empty (') or NULL and *tabname* is specified, an error is returned. To retrieve information for a specific table, the table must be identified by both schema and table name.
- If both *tabschema* and *tabname* are empty (') or NULL, information is returned for all tables.
- If *tabschema* or *tabname* do not exist, or *tabname* does not correspond to a table name (type T), a materialized query table name (type S) or a hierarchy table name (type H), an empty result set is returned.
- When the ADMIN_GET_TAB_COMPRESS_INFO_V97 table function is retrieving data for a given table, it will acquire a shared lock on the corresponding row of SYSTABLES to ensure consistency of the data that is returned (for example, to ensure that the table is not altered while information is being retrieved for it). The lock will only be held for as long as it takes to retrieve the compression information for the table, and not for the duration of the table function call.
- If the queried table is a non-XML table, there will be a row returned for the XML storage object (XDA).

ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO_V97 table function metadata

Table 73. ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO_V97 table function metadata

Column Name	Data Type	Description
TABSCHEMA	VARCHAR(128)	Schema name
TABNAME	VARCHAR(128)	Table name
DBPARTITIONNUM	SMALLINT	Database partition number
DATA_PARTITION_ID	INTEGER	Data partition number
COMPRESS_ATTR	CHAR(1)	The state of the COMPRESS attribute on the table which can be one of the following: • 'Y' = Row compression is set to yes • 'N' = Row compression is set to no
DICT_BUILDER	VARCHAR(30)	Code path taken to build the dictionary which can be one of the following: • 'INSPECT' = INSPECT ROWCOMPESTIMATE • 'LOAD' = LOAD INSERT/REPLACE • 'NOT BUILT' = no dictionary available • 'REDISTRIBUTE' = REDISTRIBUTE • 'REORG' = REORG RESETDICTIONARY • 'TABLE GROWTH' = INSERT
DICT_BUILD_TIMESTAMP	TIMESTAMP	Timestamp of when the dictionary was built. Timestamp granularity is to the second. If no dictionary is available, then the timestamp is NULL.
COMPRESS_DICT_SIZE	BIGINT	Size of compression dictionary measured in bytes.
EXPAND_DICT_SIZE	BIGINT	Size of the expansion dictionary measured in bytes. If a historical dictionary exists, this value is the sum of the current and historical dictionary sizes.
ROWS_SAMPLED	INTEGER	Number of records that contributed to building the dictionary. Migrated tables with compression dictionaries will return NULL in this column.
PAGES_SAVED_PERCENT	SMALLINT	Percentage of pages saved from compression. This information is based on the record data in the sample buffer only. Migrated tables with compression dictionaries will return NULL in this column.
BYTES_SAVED_PERCENT	SMALLINT	Percentage of bytes saved from compression. This information is based on the record data in the sample buffer only. Migrated tables with compression dictionaries will return NULL in this column.
AVG_COMPRESS_REC_LENGTH	SMALLINT	Average compressed record length of the records contributing to building the dictionary. Migrated tables with compression dictionaries will return NULL in this column.
OBJECT_TYPE	VARCHAR(4)	The type of the object. Depending on the type, this row contains values pertaining to the specified object. Values can be one of the following: • 'XML' • 'DATA'

ADMINTABINFO administrative view and ADMIN_GET_TAB_INFO_V97 table function - retrieve table size and state information

The ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V97 table function provide methods to retrieve table size and state information that is not currently available in the catalog views.

ADMINTABINFO administrative view

The ADMINTABINFO administrative view returns size and state information for tables, materialized query tables (MQT) and hierarchy tables only. These table types are reported as T for table, S for materialized query tables and H for hierarchy tables in the SYSCAT.TABLES catalog view. The information is returned at both the data partition level and the database partition level for a table.

The schema is SYSIBMADM.

Refer to the ADMINTABINFO administrative view and ADMIN_GET_TAB_INFO_V97 table function metadata table for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the ADMINTABINFO administrative view
- CONTROL privilege on the ADMINTABINFO administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the ADMIN_GET_TAB_INFO_V97 table function
- DATAACCESS authority

Examples

Example 1: Retrieve size and state information for all tables

```
SELECT * FROM SYSIBMADM.ADMINTABINFO
```

Example 2: Determine the amount of physical space used by a large number of sparsely populated tables.

```
SELECT TABSCHEMA, TABNAME, SUM(DATA_OBJECT_P_SIZE),
   SUM(INDEX_OBJECT_P_SIZE), SUM(LONG_OBJECT_P_SIZE),
   SUM(LOB_OBJECT_P_SIZE), SUM(XML_OBJECT_P_SIZE)
   FROM SYSIBMADM.ADMINTABINFO GROUP BY TABSCHEMA, TABNAME
```

Example 3: Identify tables that are eligible to use large RIDs, but are not currently enabled to use large RIDs.

```
SELECT TABSCHEMA, TABNAME FROM SYSIBMADM.ADMINTABINFO
   WHERE LARGE_RIDS = 'P'
```

Example 4: Identify which tables are using type-1 indexes and require a reorganization to convert to type-2 indexes.

```
SELECT TABSCHEMA, TABNAME FROM SYSIBMADM.ADMINTABINFO
   WHERE INDEX_TYPE = 1
```

Example 5: Identify which tables have XML data in type-1 format and require an online table move to convert to type-2 format.

```
SELECT TABSCHEMA, TABNAME FROM SYSIBMADM.ADMINTABINFO
   WHERE XML_RECORD_TYPE=1
```

Example 4: Check the current type of statistics information collected for table T1

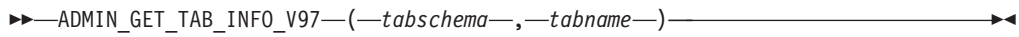
```
SELECT SUBSTR(TABSCHEMA, 1, 10) AS TBSCHEMA, SUBSTR(TABNAME, 1, 10)
   AS TBNAME, STATSTYPE FROM SYSIBMADM.ADMINTABINFO WHERE TABNAME = 'T1';

TBSCHEMA   TBNAME     STATSTYPE
---------- ---------- ---------
DB2USER1   T1         U

  1 record(s) selected.
```

ADMIN_GET_TAB_INFO_V97 table function

The ADMIN_GET_TAB_INFO_V97 table function returns the same information as the ADMINTABINFO administrative view, but allows you to specify a schema and table name.

Refer to the ADMINTABINFO administrative view and ADMIN_GET_TAB_INFO_V97 table function metadata table for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

tabschema
An input argument of type VARCHAR(128) that specifies a schema name.

tabname
An input argument of type VARCHAR(128) that specifies a table name, a materialized query table name or a hierarchy table name.

Authorization

EXECUTE privilege on the ADMIN_GET_TAB_INFO_V97 table function.

Examples

Example 1: Retrieve size and state information for the table DBUSER1.EMPLOYEE.

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_TAB_INFO_V97('DBUSER1', 'EMPLOYEE'))
   AS T
```

Example 2: Suppose there exists a non-partitioned table (DBUSER1.EMPLOYEE), with all associated objects (for example, indexes and LOBs) stored in a single table space. Calculate how much physical space the table is using in the table space:

```
SELECT (data_object_p_size + index_object_p_size + long_object_p_size +
   lob_object_p_size + xml_object_p_size) as total_p_size
   FROM TABLE( SYSPROC.ADMIN_GET_TAB_INFO_V97( 'DBUSER1', 'EMPLOYEE' )) AS T
```

Calculate how much space would be required if the table were moved to another table space, where the new table space has the same page size and extent size as the original table space:

```
SELECT (data_object_l_size + index_object_l_size + long_object_l_size +
   lob_object_l_size + xml_object_l_size) as total_l_size
   FROM TABLE( SYSPROC.ADMIN_GET_TAB_INFO_V97( 'DBUSER1', 'EMPLOYEE' )) AS T
```

Example 3: Determine the total size for the compression dictionaries for the table DBUSER1.EMPLOYEE.

```
SELECT SUBSTR(TABSCHEMA,1,10) AS TBSCHEMA, SUBSTR(TABNAME,1,10) AS TBNAME,
   DICTIONARY_SIZE + XML_DICTIONARY_SIZE AS TOTAL_DICTIONARY_SIZE
   FROM TABLE(SYSPROC.ADMIN_GET_TAB_INFO_V97('DBUSER1','EMPLOYEE'))
```

Example 4: Determine the amount of space reclaimable from a multidimensional clustering table SAMPLE.STAFF:

```
SELECT RECLAIMABLE_SPACE
   FROM TABLE(SYSPROC.ADMIN_GET_TAB_INFO_V97('SAMPLE','STAFF'))
```

Usage notes

- If both the *tabschema* and *tabname* are specified, information is returned for that specific table only.
- If the *tabschema* is specified but *tabname* is NULL or the empty string (''), then information is returned for all tables in the given schema.
- If the *tabschema* is NULL or the empty string ('') and *tabname* is specified, then an error is returned. To retrieve information for a specific table, the table must be identified by both schema and table name.
- If both *tabschema* and *tabname* are NULL or the empty string (''), then information is returned for all tables.
- If *tabschema* or *tabname* do not exist, or *tabname* does not correspond to a table name (type T), a materialized query table name (type S) or a hierarchy table name (type H), an empty result set is returned.
- When the ADMIN_GET_TAB_INFO_V97 table function is retrieving data for a given table, it will acquire a shared lock on the corresponding row of SYSTABLES to ensure consistency of the data that is returned (for example, to ensure that the table is not dropped while information is being retrieved for it). The lock will only be held for as long as it takes to retrieve the size and state information for the table, not for the duration of the table function call.
- Physical size reported for tables in SMS table spaces is the same as logical size.
- When an inplace reorg is active on a table, the physical size for the data object (DATA_OBJECT_P_SIZE) will not be calculated. Only the logical size will be returned. You can tell if an inplace reorg is active on the table by looking at the INPLACE_REORG_STATUS output column.
- The logical size reported for LOB objects created before DB2 UDB Version 8 might be larger than the physical size if the objects have not yet been reorganized.

REDISTRIBUTING_PENDING

1. no redistribute has been run for the given table N
2. redistribute started to run on the database partition group but not on the table N
3. redistribute failed in the phase before moving data N
4. redistribute failed in the phase of moving data Y
5. redistribute completely successfully and committed for the table N

ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V97 table function metadata

Table 74. ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V97 table function metadata

Column name	Data type	Description
TABSCHEMA	VARCHAR(128)	Schema name.
TABNAME	VARCHAR(128)	Table name.
TABTYPE	CHAR(1)	Table type: • 'H' = hierarchy table • 'S' = materialized query table • 'T' = table
DBPARTITIONNUM	SMALLINT	Database partition number.
DATA_PARTITION_ID	INTEGER	Data partition number.
AVAILABLE	CHAR(1)	State of the table: • 'N' = the table is unavailable. If the table is unavailable, all other output columns relating to the size and state will be NULL. • 'Y' = the table is available. **Note:** Rollforward through an unrecoverable load will put a table into the unavailable state.
DATA_OBJECT_L_SIZE	BIGINT	Data object logical size. Amount of disk space logically allocated for the table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated for the table (for example, in the case of a logical table truncation). For multi-dimensional clustering (MDC) tables, this size includes the logical size of the block map object. The size returned takes into account full extents that are logically allocated for the table and, for objects created in DMS table spaces, an estimate of the Extent Map Page (EMP) extents. This size represents the logical size of the base table only. Space consumed by LOB data, Long Data, Indexes and XML objects are reported by other columns.
DATA_OBJECT_P_SIZE	BIGINT	Data object physical size. Amount of disk space physically allocated for the table, reported in kilobytes. For MDC tables, this size includes the size of the block map object. The size returned takes into account full extents allocated for the table and includes the EMP extents for objects created in DMS table spaces. This size represents the physical size of the base table only. Space consumed by LOB data, Long Data, Indexes and XML objects are reported by other columns.

Table 74. ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V97 table function metadata (continued)

Column name	Data type	Description
INDEX_OBJECT_L_SIZE	BIGINT	Index object logical size. Amount of disk space logically allocated for the indexes defined on the table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold index data for the table (for example, in the case of a logical table truncation). The size returned takes into account full extents that are logically allocated for the indexes and, for indexes created in DMS table spaces, an estimate of the EMP extents. For partitioned indexes on partitioned tables, this is the logical size of the index object containing index partitions for the data partition identified by DATA_PARTITION_ID. This value does not take into account nonpartitioned indexes on partitioned tables. For information on both partitioned and nonpartitioned indexes, you can use the ADMIN_GET_INDEX_INFO function.
INDEX_OBJECT_P_SIZE	BIGINT	Index object physical size. Amount of disk space physically allocated for the indexes defined on the table, reported in kilobytes. The size returned takes into account full extents allocated for the indexes and includes the EMP extents for indexes created in DMS table spaces. For partitioned indexes on partitioned tables, this is the physical size of the index object containing index partitions for the data partition identified by DATA_PARTITION_ID. This value does not take into account nonpartitioned indexes on partitioned tables. For information on both partitioned and nonpartitioned indexes, you can use the ADMIN_GET_INDEX_INFO function.
LONG_OBJECT_L_SIZE	BIGINT	Long object logical size. Amount of disk space logically allocated for long field data in a table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold long field data for the table (for example, in the case of a logical table truncation). The size returned takes into account full extents that are logically allocated for long field data and, for long field data created in DMS table spaces, an estimate of the EMP extents.
LONG_OBJECT_P_SIZE	BIGINT	Long object physical size. Amount of disk space physically allocated for long field data in a table, reported in kilobytes. The size returned takes into account full extents allocated for long field data and includes the EMP extents for long field data created in DMS table spaces.

Table 74. ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V97 table function metadata (continued)

Column name	Data type	Description
LOB_OBJECT_L_SIZE	BIGINT	LOB object logical size. Amount of disk space logically allocated for LOB data in a table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold LOB data for the table (for example, in the case of a logical table truncation). The size includes space logically allocated for the LOB allocation object. The size returned takes into account full extents that are logically allocated for LOB data and, for LOB data created in DMS table spaces, an estimate of the EMP extents.
LOB_OBJECT_P_SIZE	BIGINT	LOB object physical size. Amount of disk space physically allocated for LOB data in a table, reported in kilobytes. The size includes space allocated for the LOB allocation object. The size returned takes into account full extents allocated for LOB data and includes the EMP extents for LOB data created in DMS table spaces.
XML_OBJECT_L_SIZE	BIGINT	XML object logical size. Amount of disk space logically allocated for XML data in a table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold XML data for the table (for example, in the case of a logical table truncation). The size returned takes into account full extents that are logically allocated for XML data and, for XML data created in DMS table spaces, an estimate of the EMP extents.
XML_OBJECT_P_SIZE	BIGINT	XML object physical size. Amount of disk space physically allocated for XML data in a table, reported in kilobytes. The size returned takes into account full extents allocated for XML data and includes the EMP extents for XML data created in DMS table spaces.
INDEX_TYPE	SMALLINT	Indicates the type of indexes currently in use for the table. Returns: • 1 if type-1 indexes are being used. • 2 if type-2 indexes are being used.
REORG_PENDING	CHAR(1)	A value of 'Y' indicates that a reorg recommended alter has been applied to the table and a classic (offline) reorg is required. Otherwise 'N' is returned.
INPLACE_REORG_STATUS	VARCHAR(10)	Current status of an inplace table reorganization on the table. The status value can be one of the following: • ABORTED (in a PAUSED state, but unable to RESUME; STOP is required) • EXECUTING • NULL (if no inplace reorg has been performed on the table) • PAUSED

Table 74. ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V97 table function metadata (continued)

Column name	Data type	Description
LOAD_STATUS	VARCHAR(12)	Current status of a load operation against the table. The status value can be one of the following: • IN_PROGRESS • NULL (if there is no load in progress for the table and the table is not in load pending state) • PENDING
READ_ACCESS_ONLY	CHAR(1)	'Y' if the table is in Read Access Only state, 'N' otherwise. A value of 'N' should not be interpreted as meaning that the table is fully accessible. If a load is in progress or pending, a value of 'Y' means the table data is available for read access, and a value of 'N' means the table is inaccessible. Similarly, if the table status is set integrity pending (refer to SYSCAT.TABLES STATUS column), then a value of 'N' means the table is inaccessible.
NO_LOAD_RESTART	CHAR(1)	A value of 'Y' indicates the table is in a partially loaded state that will not allow a load restart. A value of 'N' is returned otherwise.
NUM_REORG_REC_ALTERS	SMALLINT	Number of reorg recommend alter operations (for example, alter operations after which a reorganization is required) that have been performed against this table since the last reorganization.
INDEXES_REQUIRE_REBUILD	CHAR(1)	For nonpartitioned tables, 'Y' if any of the indexes defined on the table require a rebuild, and 'N' otherwise. For partitioned tables, 'Y' if any index partitions for the data partition identified by DATA_PARTITION_ID require a rebuild, and 'N' otherwise.
LARGE_RIDS	CHAR(1)	Indicates whether or not the table is using large row IDs (RIDs) (4 byte page number, 2 byte slot number). A value of 'Y' indicates that the table is using large RIDs and 'N' indicates that it is not using large RIDs. A value of 'P' (pending) will be returned if the table supports large RIDs (that is, the table is in a large table space), but at least one of the indexes for the table has not been reorganized or rebuilt yet, so the table is still using 4 byte RIDs (which means that action must be taken to convert the table or indexes).
LARGE_SLOTS	CHAR(1)	Indicates whether or not the table is using large slots (which allows more than 255 rows per page). A value of 'Y' indicates that the table is using large slots and 'N' indicates that it is not using large slots. A value of 'P' (pending) will be returned if the table supports large slots (that is, the table is in a large table space), but there has been no offline table reorganization or table truncation operation performed on the table yet, so it is still using a maximum of 255 rows per page.
DICTIONARY_SIZE	BIGINT	Size of the table dictionary, in bytes, used for row compression if a row compression dictionary exists for the table. If a historical dictionary exists, this value is the sum of the current and historical dictionary sizes.
BLOCKS_PENDING_CLEANUP	BIGINT	For MDC tables, the number of blocks pending cleanup. For non MDC tables this value will always be zero.

Table 74. ADMINTABINFO administrative view and the ADMIN_GET_TAB_INFO_V97 table function metadata (continued)

Column name	Data type	Description
STATSTYPE	CHAR(1)	• 'F' = System fabricated statistics without table or index scan. These statistics are stored in memory and are different from what is stored in the system catalogs. This is a temporary state and eventually full statistics will be gathered by DB2 and stored in the system catalogs. • 'A'= System asynchronously gathered statistics. Statistics have been automatically collected by DB2 by a background process and stored in the system catalogs. • 'S' = System synchronously gathered statistics. Statistics have been automatically collected by DB2 during SQL statement compilation. These statistics are stored in memory and are different from what is stored in the system catalogs. This is a temporary state and eventually DB2 will store the statistics in the system catalogs. • 'U' = User gathered statistics. Statistics gathering was initiated by the user through a utility such as RUNSTATS, CREATE INDEX, LOAD, REDISTRIBUTE or by manually updating system catalog statistics. • NULL = unknown type
XML_RECORD_TYPE	SMALLINT	Indicates the type of XML record currently in use for the table. • 1 if the type-1 (single node) XML record format is being used. • 2 if the type-2 (multi-node) XML record format is being used. • Null if the table has no XML columns.
RECLAIMABLE_SPACE	BIGINT	For an MDC table in a DMS table space, this value indicates the amount of disk space that can be reclaimed by running the REORG command with the RECLAIM option. Disk space is reported in kilobytes. For any other table, the value is zero.
XML_DICTIONARY_SIZE	BIGINT	Size of the XML dictionary, in bytes, used for data compression if a data compression dictionary exists for the XML storage object. If the table does not contain any XML columns or if a compression dictionary has not been created, the value is 0.

ADMINTEMPCOLUMNS administrative view and ADMIN_GET_TEMP_COLUMNS table function - Retrieve column information for temporary tables

The ADMINTEMPCOLUMNS administrative view and the ADMIN_GET_TEMP_COLUMNS table function provide methods to retrieve column attribute information for created temporary tables and declared temporary tables.

Although the catalog views contain column attribute information for instances of created temporary tables, they do not have this information for declared temporary tables.

ADMINTEMPCOLUMNS administrative view

The ADMINTEMPCOLUMNS administrative view returns column attribute information for instances of created temporary tables and declared temporary tables.

The schema is SYSIBMADM.

Refer to the ADMINTEMPCOLUMNS administrative view and ADMIN_GET_TEMP_COLUMNS table function metadata table for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the ADMINTEMPCOLUMNS administrative view
- CONTROL privilege on the ADMINTEMPCOLUMNS administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the ADMIN_GET_TEMP_COLUMNS table function
- DATAACCESS authority

Examples

Example 1: Retrieve column attribute information for all instances of created temporary tables and all declared temporary tables currently existing in the database.

```
SELECT * FROM SYSIBMADM.ADMINTEMPCOLUMNS
```

Example 2: Determine which temporary tables active in the database are using the user-defined data type of USMONEY.

```
SELECT APPLICATION_HANDLE, TABSCHEMA, TABNAME
   FROM SYSIBMADM.ADMINTEMPCOLUMNS
   WHERE TYPENAME = 'USMONEY'
```

Example 3: Retrieve table schema, table name, and the column names for all declared temporary tables declared by the SYSTEM_USER.

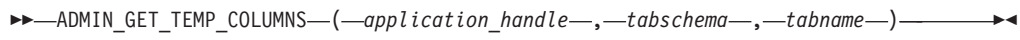
```
SELECT T.TABSCHEMA, T.TABNAME, C.COLNAME
   FROM SYSIBMADM.ADMINTEMPCOLUMNS C, SYSIBMADM.ADMINTEMPTABLES T
   WHERE T.TEMPTABTYPE = 'D'
   AND T.INSTANTIATOR = SYSTEM_USER
   AND T.TABSCHEMA = C.TABSCHEMA
   AND T.TABNAME = C.TABNAME
```

ADMIN_GET_TEMP_COLUMNS table function

The ADMIN_GET_TEMP_TABLES table function returns the same information as the ADMINTEMPCOLUMNS administrative view, but allows you to specify a schema name and a table name.

Refer to the ADMINTEMPCOLUMNS administrative view and ADMIN_GET_TEMP_COLUMNS table function metadata table for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies an application handle. If *application_handle* is specified, data is returned for the specified connection only; if *application_handle* is NULL, data is returned for all connections.

tabschema
An input argument of type VARCHAR(128) that specifies a schema name.

tabname
An input argument of type VARCHAR(128) that specifies a created temporary table name or a declared temporary table name.

Authorization

EXECUTE privilege on the ADMIN_GET_TEMP_COLUMNS table function.

Examples

Example 1: Retrieve column information for the declared temporary table TEMPEMPLOYEE for the current connection.

```
SELECT *
   FROM TABLE (
      SYSPROC.ADMIN_GET_TEMP_COLUMNS(
         APPLICATION_ID(), 'SESSION', 'TEMPEMPLOYEE'))
   AS T
```

Usage notes

- If both *tabschema* and *tabname* are specified, then information is returned for that specific temporary table only.
- If *tabschema* is specified but *tabname* is NULL or the empty string (''), then information is returned for all tables in the given schema.
- If *tabschema* is NULL or the empty string ('') and *tabname* is specified, then an error is returned. To retrieve information for a specific temporary table, the table must be identified by both schema and table name.
- If both *tabschema* and *tabname* are NULL or the empty string (''), then information is returned for all temporary tables for the connection or all connections, depending on the value of *application_handle*.
- If *tabschema* or *tabname* do not exist, or *tabname* does not correspond to a temporary table name, or instances of the identified temporary table do not exist in the database, then an empty result set is returned.

ADMINTEMPCOLUMNS administrative view and the ADMIN_GET_TEMP_COLUMNS table function metadata

Table 75. ADMINTEMPCOLUMNS administrative view and the ADMIN_GET_TEMP_COLUMNS table function metadata

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier is the same on every partition where the application can make a secondary connection.
APPLICATION_NAME	VARCHAR(256)	Name of the application.
TABSCHEMA	VARCHAR(128)	Schema name of the temporary table that contains the column.
TABNAME	VARCHAR(128)	Table name of the temporary table that contains the column.
COLNAME	VARCHAR(128)	Name of the column.
COLNO	SMALLINT	Number of this column in the table (starting with 0).
TYPESCHEMA	VARCHAR(128)	Schema name of the data type for the column.
TYPENAME	VARCHAR(128)	Unqualified name of the data type for the column.
LENGTH	INTEGER	Maximum length of the data. 0 for distinct types. The LENGTH column indicates precision for DECIMAL fields, and indicates the number of bytes of storage required for decimal floating-point columns; that is, 8 and 16 for DECFLOAT(16) and DECFLOAT(34), respectively.
SCALE	SMALLINT	Scale if the column type is DECIMAL; or the number of digits of fractional seconds if the column type is TIMESTAMP; 0 otherwise.
DEFAULT	VARCHAR(254)	Default value for the column of a table expressed as a constant, special register, or cast-function appropriate for the data type of the column. Can also be the keyword NULL. Values might be converted from what was specified as a default value. For example, date and time constants are shown in ISO format, cast-function names are qualified with schema names, and identifiers are delimited. Null value if a DEFAULT clause was not specified or the column is a view column.
NULLS	CHAR(1)	Nullability attribute for the column. • "Y" = Column is nullable • "N" = Column is not nullable The value can be "N" for a view column that is derived from an expression or function. Nevertheless, such a column allows null values when the statement using the view is processed with warnings for arithmetic errors.
CODEPAGE	SMALLINT	Code page used for data in this column; 0 if the column is defined as FOR BIT DATA or is not a string type.
LOGGED	CHAR(1)	Applies only to columns whose type is LOB or distinct based on LOB; blank otherwise. • "Y" = Column is logged • "N" = Column is not logged

Table 75. ADMINTEMPCOLUMNS administrative view and the ADMIN_GET_TEMP_COLUMNS table function metadata (continued)

Column name	Data type	Description
COMPACT	CHAR(1)	Applies only to columns whose type is LOB or distinct based on LOB; blank otherwise. • "Y" = Column is compacted in storage • "N" = Column is not compacted
INLINE_LENGTH	INTEGER	Maximum size in bytes of the internal representation of an instance of an XML document or a structured type that can be stored in the base table; 0 when not applicable.
IDENTITY	CHAR(1)	• "Y" = Identity column • "N" = Not an identity column
GENERATED	CHAR(1)	Type of generated column. • "A" = Column value is always generated • "D" = Column values is generated by default • Blank = Column is not generated

ADMINTEMPTABLES administrative view and ADMIN_GET_TEMP_TABLES table function - Retrieve information for temporary tables

The ADMINTEMPTABLES administrative view and the ADMIN_GET_TEMP_TABLES table function provide methods to retrieve table attribute and instantiation time information for instances of created temporary tables and declared temporary tables.

Although the catalog views contain table attribute information for created temporary tables, they do not contain this information for declared temporary tables. In addition, the catalog views do not contain table instantiation time information for created temporary tables or declared temporary tables.

ADMINTEMPTABLES administrative view

The ADMINTEMPTABLES administrative view returns table attribute and instantiation time information for instances of created temporary tables and declared temporary tables.

The schema is SYSIBMADM.

Refer to the ADMINTEMPTABLES administrative view and ADMIN_GET_TEMP_TABLES table function metadata table for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the ADMINTEMPTABLES administrative view
- CONTROL privilege on the ADMINTEMPTABLES administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the ADMIN_GET_TEMP_TABLES table function
- DATAACCESS authority

Examples

Example 1: Retrieve table attributes and instantiation time information for all instances of created temporary tables and declared temporary tables currently existing in the database.

```
SELECT * FROM SYSIBMADM.ADMINTEMPTABLES
```

Example 2: Determine which connections have an instance of a created temporary table.

```
SELECT APPLICATION_HANDLE, TABSCHEMA, TABNAME
   FROM SYSIBMADM.ADMINTEMPTABLES
   WHERE TEMPTABTYPE = 'C'
```

Example 3: Retrieve table attributes and instantiation time information for all declared temporary tables declared for all the tables instantiated by the user that connected to the database.

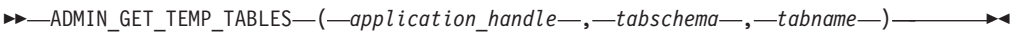
```
SELECT TABSCHEMA, TABNAME, ONCOMMIT, ONROLLBACK,
      INSTANTIATION_TIME
   FROM SYSIBMADM.ADMINTEMPTABLES
   WHERE TEMPTABTYPE = 'D' AND INSTANTIATOR = SYSTEM_USER
```

ADMIN_GET_TEMP_TABLES table function

The ADMIN_GET_TEMP_TABLES table function returns the same information as the ADMINTABINFO administrative view, but allows you to specify a schema name and a table name.

Refer to the ADMINTABINFO administrative view and ADMIN_GET_TEMP_TABLES table function metadata table for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies an application handle. If *application_handle* is specified, data is returned for the specified connection only; if *application_handle* is NULL, data is returned for all connections.

tabschema
An input argument of type VARCHAR(128) that specifies a schema name.

tabname
An input argument of type VARCHAR(128) that specifies a created temporary table name or a declared temporary table name.

Authorization

EXECUTE privilege on the ADMIN_GET_TEMP_TABLES table function.

Examples

Example 1: Retrieve table attributes and instantiation time information for all instances of the created temporary table DBUSER1.EMPLOYEE for all connections.

```
SELECT TABSCHEMA, TABNAME, ONCOMMIT, ONROLLBACK, INSTANTIATION_TIME
   FROM TABLE (SYSPROC.ADMIN_GET_TEMP_TABLES(NULL, 'DBUSER1', 'EMPLOYEE'))
   AS T
```

Example 2: Retrieve the instantiation time and table space ID for all instances of user temporary tables for the current connection.

```
SELECT TABSCHEMA, TABNAME, INSTANTIATION_TIME, TBSP_ID
   FROM TABLE (SYSPROC.ADMIN_GET_TEMP_TABLES(APPLICATION_ID(), '', ''))
   AS T
```

Usage notes

•

- If both *tabschema* and *tabname* are specified, then information is returned for that specific temporary table only.
- If *tabschema* is specified but *tabname* is NULL or the empty string (''), then information is returned for all tables in the given schema.
- If *tabschema* is NULL or the empty string ('') and *tabname* is specified, then an error is returned. To retrieve information for a specific temporary table, the table must be identified by both schema and table name.
- If both *tabschema* and *tabname* are NULL or the empty string (''), then information is returned for all temporary tables for the connection or all connections, depending on the value of *application_handle*.
- If *tabschema* or *tabname* do not exist, or *tabname* does not correspond to a temporary table name, or instances of the identified temporary table do not exist in the database, then an empty result set is returned.

ADMINTEMPTABLES administrative view and the ADMIN_GET_TEMP_TABLES table function metadata

Table 76. ADMINTEMPTABLES administrative view and the ADMIN_GET_TEMP_TABLES table function metadata

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier is the same on every partition where the application can make a secondary connection.
APPLICATION_NAME	VARCHAR(256)	Name of the application.
TABSCHEMA	VARCHAR(128)	Schema name.
TABNAME	VARCHAR(128)	Table name.
INSTANTIATOR	VARCHAR(128)	Authorization ID under which the created temporary table was instantiated or declared temporary table was declared.
INSTANTIATORTYPE	CHAR(1)	• "U" = The instantiator is an individual user
TEMPTABTYPE	CHAR(1)	Temporary table type: • "C" = Created temporary table • "D" = Declared temporary table

Table 76. ADMINTEMPTABLES administrative view and the ADMIN_GET_TEMP_TABLES table function metadata (continued)

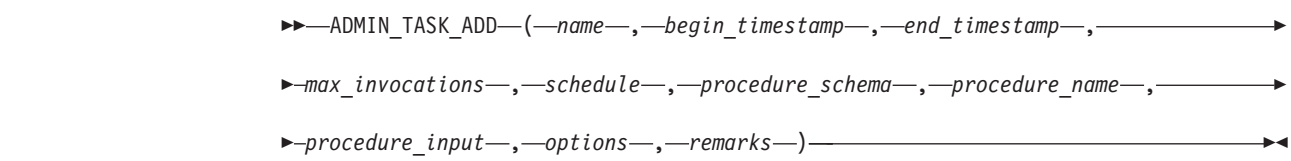
Column name	Data type	Description
INSTANTIATION_TIME	TIMESTAMP	Time at which the created temporary table instance was instantiated or the declared temporary table was declared.
COLCOUNT	SMALLINT	Number of columns, including inherited columns (if any).
TAB_FILE_ID	BIGINT	table_file_id - The file ID (FID) for the table.
TBSP_ID	BIGINT	tablespace_id - An integer that uniquely represents a table space used by the current database.
PMAP_ID	SMALLINT	Identifier for the distribution map that is currently in use by this table.
PARTITION_MODE	CHAR(1)	Indicates how data is distributed among database partitions in a partitioned database system. • "H" = Hashing • Blank = No database partitioning
CODEPAGE	SMALLINT	Code page of the object. This is the default code page used for all character columns and expression-generated columns.
ONCOMMIT	CHAR(1)	Specifies the action taken on this table when a COMMIT operation is performed. • "D" = Delete rows • "P" = Preserve rows
ONROLLBACK	CHAR(1)	Specifies the action taken on this table when a ROLLBACK operation is performed. • "D" = Delete rows • "P" = Preserve rows
LOGGED	CHAR(1)	Specifies whether this table is logged. • "N" = Not logged • "Y" = Logged

Chapter 5. Administrative Task Scheduler routines and views

ADMIN_TASK_ADD procedure - Schedule a new task

The ADMIN_TASK_ADD procedure schedules an administrative task, which is any piece of work that can be encapsulated inside a procedure.

Syntax



The schema is SYSPROC.

Procedure parameters

name
An input argument of type VARCHAR(128) that specifies the name of the task. This argument cannot be NULL.

begin_timestamp
An input argument of type TIMESTAMP that specifies the earliest time a task can begin execution. The value of this argument cannot be in the past, and it cannot be later than *end_timestamp*.

When task execution begins depends on how this argument and the *schedule* argument are defined:

- If the *begin_timestamp* argument is not NULL:
  - If the *schedule* argument is NULL, the task execution begins at *begin_timestamp*.
  - If the *schedule* argument is not NULL, the task execution begins at the next scheduled time at or after *begin_timestamp*.
- If the *begin_timestamp* argument is NULL:
  - If the *schedule* argument is NULL, the task execution begins immediately.
  - If the *schedule* argument is not NULL, the task execution begins at the next scheduled time.

end_timestamp
An input argument of type TIMESTAMP that specifies the last time that a task can begin execution. The value of this argument cannot be in the past, and it cannot be earlier than *begin_timestamp*. If the argument is NULL, the task can continue to execute as scheduled indefinitely.

An executing task will not be interrupted at its *end_timestamp*.

max_invocations
An input argument of type INTEGER that specifies the maximum number of executions allowed for the task. If the argument is NULL, there is no limit to the number of times the task can execute. If the argument is 0, the task will not execute.

This value applies to the schedule if *schedule* is not NULL.

If both *end_timestamp* and *max_invocations* are specified, *end_timestamp* takes precedence. That is, if the *end_timestamp* timestamp is reached, even though the number of task executions so far has not reached the value of *max_invocations*, the task will not be executed again.

schedule
An input argument of type VARCHAR(1024) that specifies a task execution schedule at fixed points in time. If the argument is NULL, the task is not scheduled at fixed points in time.

The *schedule* string must be specified using the UNIX cron format.

Multiple schedules are not supported.

procedure_schema
An input argument of type VARCHAR(128) that specifies the schema of the procedure that this task will execute. This argument cannot be NULL.

procedure_name
An input argument of type VARCHAR(128) that specifies the name of the procedure that this task will execute. This argument cannot be NULL.

procedure_input
An input argument of type CLOB(2M) that specifies the input arguments of the procedure that this task will execute. This argument must contain an SQL statement that returns one row of data. The returned values will be passed as arguments to the procedure. If this argument is NULL, no arguments are passed to the procedure.

The number of columns returned by the SQL statement must match the total number (and type) of arguments for the procedure and must contain a single row. For output arguments, the value itself is ignored, but should be of the same SQL data type as the procedure requires.

This SQL statement is executed every time the task is executed. If the SQL statement fails, the task's status will be set to NOTRUN and specific SQLCODE information will be recorded. If the statement does not return a result set, does not return a row, returns multiple rows or result sets the task will not be executed. The task's status will be set to NOTRUN and SQLCODE SQL1465N will be set to indicate that this argument is invalid.

If the statement result contains serialized XML parameters, the total size of all XML parameters combined is limited to 256 kilobytes. If the result exceeds this threshold, the task's status will be set to NOTRUN. SQLCODE -302 and SQLSTATE 22001 will be set to indicate that data truncation has occurred.

To view the task's status, use the SYSTOOL.ADMIN_TASK_STATUS view

options
An input argument of type VARCHAR(512). This argument must be NULL.

remarks
An input argument of type VARCHAR(254) that specifies a description of the task. This argument is optional and can be NULL.

Authorization

EXECUTE privilege on the ADMIN_TASK_ADD procedure. Unless the database was created with the **RESTRICTIVE** option, EXECUTE privilege is granted to PUBLIC by default.

Usage notes

The SYSTOOLSPACE table space must exist before you call the ADMIN_TASK_ADD procedure. If it does not exist, the procedure will return an SQL0204N error message.

When a task is scheduled, the authorization ID of the current session user is recorded. The scheduler switches to this session authorization ID when the executing the task.

The administrative task scheduler does not support the execution of procedures that perform a database connection without a specified user ID and password. For example, the ADMIN_CMD procedure can be used to perform a LOAD from a database. A connection to the source database is established using the user ID and password provided for the currently connected database. This type of LOAD operation cannot be executed by the task scheduler.

If invalid arguments are passed into the procedure, SQL0171N will be returned. The tokens of the message will indicate which argument is invalid and the name of the procedure.

The task cannot be scheduled for execution until the unit of work is committed and the scheduler has fetched the task definition.

The scheduler checks for new or updated tasks every 5 minutes. To ensure the task executes as expected, the earliest begin time, as defined by the *begin_timestamp*, *end_timestamp* and *schedule* arguments, should be at least 5 minutes after the unit of work commits.

The database must be active on all database partitions to ensure the task can be executed by the scheduler.

In a partitioned database environment, the ADMIN_TASK_ADD procedure can be called from any database partition. The scheduler, however, will execute all tasks from the catalog database partition.

The *begin_timestamp*, *end_timestamp*, and *schedule* are based on the server's time zone. Special attention is required when scheduling a task during the transition period of daylight savings time (DST). If the task is scheduled to run 2:01 AM and it is the time of year when the time springs forward, the task will not run as the clock skips from 2:00 AM to 3:00 AM. On the other hand, when the time falls back an hour, tasks that were originally scheduled between 2:00 AM and 3:00 AM will execute twice. The user is responsible for making adjustments for daylight savings time to ensure their desired behavior.

The scheduler will always commit after calling the procedure specified by *procedure_schema* and *procedure_name*. If a transaction roll back is required, the rollback must occur inside the procedure.

If the task name is not unique, the procedure will fail with SQL0601N.

Example

Example 1: Create a task that performs an online TSM backup daily at 12:00 AM, starting on February 4, 2008:

```
CALL SYSPROC.ADMIN_TASK_ADD
 ( 'DAILY TSM BACKUP',
     '2007-02-04-00.00.00.000000',
      NULL,
       NULL,
       '0 0 * * *',
       'SYSPROC',
       'ADMIN_CMD',
       'VALUES(''BACKUP DATABASE SALES ONLINE USE TSM WITHOUT PROMPTING'')',
       NULL,
       NULL )
```

Example 2: Schedule a task to flush an event monitor every hour:

1. Create an SQL procedure, in the PROD schema, that flushes an event monitor called "em":

   ```
   CREATE PROCEDURE FLUSH_EVENT_MONITOR()
    SPECIFIC FLUSH_EVENT_MONITOR
      LANGUAGE SQL
      BEGIN
      DECLARE stmt VARCHAR(100) ;
      SET stmt = 'FLUSH EVENT MONITOR em' ;
      EXECUTE IMMEDIATE stmt ;
      END
   ```

   Note: The FLUSH EVENT MONITOR SQL statement cannot be called directly in the procedure. However, EXECUTE IMMEDIATE can be used.

2. Call ADMIN_TASK_ADD to schedule the task:

   ```
   CALL SYSPROC.ADMIN_TASK_ADD
    ('FLUSH EVENT MONITOR EVERY HOUR',
      NULL,
       NULL,
      NULL,
      '0 0-23 * * *',
      'PROD',
      'FLUSH_EVENT_MONITOR',
      NULL,
      NULL,
      NULL )
   ```

UNIX cron format

The UNIX cron format is used to specify time in the *schedule* parameter of the ADMIN_TASK_ADD and ADMIN_TASK_UPDATE procedures.

The cron format has five time and date fields separated by at least one blank. There can be no blank within a field value. Scheduled tasks are executed when the *minute*, *hour*, and *month of year* fields match the current time and date, and at least one of the two day fields (*day of month*, or *day of week*) match the current date.

Table 1 lists the time and date fields and their allowed values in cron format.

Table 77. Field names and values for the UNIX cron format

Field name	Allowed values
minute	0-59
hour	0-23
day of month	1-31

Table 77. Field names and values for the UNIX cron format (continued)

Field name	Allowed values
month	• 1-12, where 1 is January, 2 is February, and so on. • Uppercase, lowercase and mixed-case three character strings, based on the English name of the month. For example: jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, or dec.
day of week	• 0-7, where 0 or 7 is Sunday, 1 is Monday, and so on. • Uppercase, and lowercase or mixed-case three character strings, based on the English name of the day: mon, tue, wed, thu, fri, sat, or sun.

Ranges and lists

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, the range 8-11 for an hour entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers or ranges separated by commas. For example:

```
1,2,5,9
0-4,8-12
```

Unrestricted range

A field can contain an asterisk (*), which represents all possible values in the field.

The day of a command's execution can be specified by two fields: *day of month* and *day of week*. If both fields are restricted by the use of a value other than the asterisk, the command will run when either field matches the current time. For example, the value `30 4 1,15 * 5` causes a command to run at 4:30 AM on the 1st and 15th of each month, plus every Friday.

Step values

Step values can be used in conjunction with ranges. The syntax *`range/step`* defines the range and an execution interval.

If you specify *`first-last/step`*, execution takes place at *first*, then at all successive values that are distant from *first* by *step*, until *last*.

For example, to specify command execution every other hour, use `0-23/2`. This expression is equivalent to the value `0,2,4,6,8,10,12,14,16,18,20,22`.

If you specify `*/step`, execution takes place at every interval of *step* through the unrestricted range. For example, as an alternative to `0-23/2` for execution every other hour, use `*/2`.

Example

Table 2 lists values that you can use for the *schedule* argument in ADMIN_TASK_ADD or ADMIN_TASK_UPDATE procedures for various scheduling scenarios.

Table 78. Example task schedules and the appropriate schedule argument values

Desired task schedule	*schedule* **value**
2:10 PM every Monday	`10 14 * * 1`
Every day at midnight	`0 0 * * *`
Every weekday at midnight	`0 0 * * 1-5`
Midnight on 1st and 15th day of the month	`0 0 1,15 * *`
6.32 PM on the 17th, 21st and 29th of November plus each Monday and Wednesday in November each year	`32 18 17,21,29 11 mon,wed`

ADMIN_TASK_LIST administrative view - Retrieve information about tasks in the scheduler

The ADMIN_TASK_LIST administrative view retrieves information about each task defined in the administrative task scheduler.

The schema is SYSTOOLS.

This view is created the first time the ADMIN_TASK_ADD procedure is called.

Authorization

SELECT or CONTROL privilege on the ADMIN_TASK_LIST administrative view. Unless the database was created with the **RESTRICTIVE** option, SELECT privilege is granted to PUBLIC by default.

When you query the ADMIN_TASK_LIST view, it will only return the tasks that were created using your session authorization ID. If you have SYSADM, SYSCTRL, SYSMAINT, or DBADM authority, all tasks are returned.

Example

Request the list of tasks in the scheduler:

```
SELECT * from SYSTOOLS.ADMIN_TASK_LIST
```

Information returned

Table 79. Information returned by the ADMIN_TASK_LIST administrative view

Column name	**Data type**	**Description**
NAME	VARCHAR(128)	The name of the task.
TASKID	INTEGER	The task identifier.
OWNER	VARCHAR(128)	The session authorization ID of the user that created the task.

Table 79. Information returned by the ADMIN_TASK_LIST administrative view (continued)

Column name	Data type	Description
OWNERTYPE	VARCHAR(1)	The authorization ID type. Valid values are: • U - User
BEGIN_TIME	TIMESTAMP	The timestamp of when the task is first able to run.[1]
END_TIME	TIMESTAMP	The timestamp of when the task is last able to run.[1] If this column is NULL, the task can run indefinitely unless MAX_INVOCATIONS is specified.
MAX_INVOCATIONS	INTEGER	The maximum number of executions allowed for the task. If this column is NULL, the task can run indefinitely unless END_TIME is specified.
SCHEDULE	VARCHAR(1024)	The schedule for the task, in UNIX cron format.
PROCEDURE_SCHEMA	VARCHAR(128)	The schema of the procedure that this task will execute.
PROCEDURE_NAME	VARCHAR(128)	The name of the procedure that this task will execute.
PROCEDURE_INPUT	CLOB(2M)	The input parameters of the procedure that this task will execute. If this column is NULL, there are no input parameters.
OPTIONS	VARCHAR(512)	Options that affect the behavior of the task.
UPDATE_TIME	TIMESTAMP	The timestamp when the task was last updated.
REMARKS	VARCHAR(254)	A description of the task.

Note:

- [1] The BEGIN_TIME and END_TIME are based on the database server's time zone. The user is responsible for making adjustments for daylight savings time (DST).

ADMIN_TASK_REMOVE procedure - Remove scheduled tasks or task status records

The ADMIN_TASK_REMOVE procedure removes scheduled administrative tasks, which are pieces of work that can be encapsulated inside a procedure. It also removes task status records.

Syntax

The schema is SYSPROC.

Procedure parameters

name
An input argument of type VARCHAR(128) that specifies the name of the task.

end_timestamp
An output argument of type TIMESTAMP that specifies the status record *end_timestamp* timestamp.

Authorization

EXECUTE privilege on the ADMIN_TASK_REMOVE procedure. Unless the database was created with the **RESTRICTIVE** option, EXECUTE privilege is granted to PUBLIC by default.

Although the statement authorization ID might allow the procedure to be executed, successful removal of task and status records depends on the value of the current session authorization ID. The current session authorization ID must match the session authorization ID that was recorded when the task was created. Users with SYSADM, SYSCTRL, SYSMAINT, or DBADM authority can remove any task or status record. If an unauthorized user attempts to remove a task or status record, an SQL0551N is returned.

Usage notes

The task is not removed until the unit of work is committed.

The behavior of the task removal depends on how the *name* and *end_timestamp* arguments are defined:

- If the *end_timestamp* argument is `NULL`:
  - If the *name* argument is `NULL`, all tasks and status records are removed. If one or more tasks are currently running, then the task and associated status records are not removed. In this case, SQL1464W is returned.
  - If the *name* argument is not `NULL`, the task record that matches *name* is removed. If the specified task is currently running, the task is not removed and SQL20453N is returned. If the specified task is removed, all associated status records are removed.
- If the *end_timestamp* argument is not `NULL`:
  - If the *name* argument is `NULL`, all status records with *end_timestamp* timestamps less than or equal to *end_timestamp* are removed. No task records are removed. The procedure will not remove any status records that have a status value of `RUNNING`.
  - If the *name* argument is not `NULL`, the status records for the task that matches *name* are removed if their *end_timestamp* timestamp is less than or equal to *end_timestamp*. No task records are removed. The procedure will not remove any status records that have a status value of `RUNNING`.

If a user attempts to remove a task that does not exist, an SQL0204N is returned.

Example

Remove a backup task called 'DAILY TSM BACKUP':

```
CALL SYSPROC.ADMIN_TASK_REMOVE('DAILY TSM BACKUP', NULL)
```

ADMIN_TASK_STATUS administrative view - Retrieve task status information

The ADMIN_TASK_STATUS administrative view retrieves information about the status of task execution in the administrative task scheduler.

The schema is SYSTOOLS.

This view is created the first time the ADMIN_TASK_ADD procedure is called.

Authorization

SELECT or CONTROL privilege on the ADMIN_TASK_STATUS administrative view. Unless the database was created with the **RESTRICTIVE** option, SELECT privilege is granted to PUBLIC by default.

When you query the ADMIN_TASK_STATUS view, it will only return the task status records that were created by your session authorization ID.

Example

Example 1: Request the status of tasks in the scheduler:

```
SELECT * from SYSTOOLS.ADMIN_TASK_STATUS
```

Example 2: Format the data in the SQLERRMC column using the SQLERRM function:

```
SELECT TASKID, STATUS, SQLCODE, SQLSTATE, RC,
   VARCHAR( SQLERRM( 'SQL' || CHAR( ABS(SQLCODE) ),
   SQLERRMC, x'FF', 'en_US', 1 ), 256) AS MSG_TXT
   FROM SYSTOOLS.ADMIN_TASK_STATUS
```

Information returned

Table 80. Information returned by the ADMIN_TASK_STATUS administrative view

Column name	Data type	Description
NAME	VARCHAR(128)	The name of the task.
TASKID	INTEGER	The task identifier.
STATUS	VARCHAR(10)	The status of the task. Valid values are: • `RUNNING` - The task is currently running. • `COMPLETED` - The task has finished running. • `NOTRUN` - An error prevented the scheduler from calling the task's procedure. • `UNKNOWN` - The task started running but an unexpected condition prevented the scheduler from recording the task outcome. This can occur if the system ends abnormally or a power failure happens while the task is running.
INVOCATION	INTEGER	The current invocation count.
BEGIN_TIME	TIMESTAMP	The time that the task began.[1] If the STATUS is `RUNNING`, `COMPLETED`, or `UNKNOWN`, this value indicates the time that the task started running. If the STATUS is `NOTRUN`, it indicates the time that the task should have started.

Table 80. Information returned by the ADMIN_TASK_STATUS administrative view (continued)

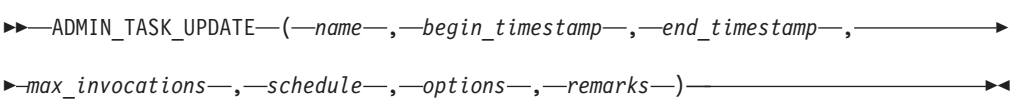
Column name	Data type	Description
END_TIME	TIMESTAMP	The time that the task finished running.[1] This value will be NULL if the STATUS is RUNNING. If the STATUS is UNKNOWN, this value is the time the task scheduler detected the task was no longer executing and updated the status table.
AGENT_ID	BIGINT	The agent ID for the application executing the task. The agent ID is synonymous with the application handle. This value is only valid while the task is executing.
SQLCODE	INTEGER	If the STATUS is COMPLETED, this value indicates the SQLCODE returned by the CALL to the procedure. If the STATUS is NOTRUN, this value indicates the SQLCODE of the error that prevented the task from running. If the status is RUNNING or UNKNOWN, this value will be NULL.
SQLSTATE	CHAR(5)	If the STATUS is COMPLETED, this value indicates the SQLSTATE returned by the CALL to the procedure. If the STATUS is NOTRUN, this value indicates the SQLSTATE of the error that prevented the task from running. If the status is RUNNING or UNKNOWN, this value will be NULL.
SQLERRMC	VARCHAR(70) FOR BIT DATA	Contains one or more tokens, separated by X'FF', as they appear in the SQLERRMC field of the SQLCA. These tokens are substituted for variables in the descriptions of error conditions If the STATUS is COMPLETED, this value indicates the SQLERRMC returned by the CALL to the procedure. If the STATUS is NOTRUN, this value indicates the SQLERRMC of the error that prevented the task from running. If the status is RUNNING or UNKNOWN, this value will be NULL.
RC	INTEGER	If the STATUS is COMPLETED, this contains the return code from the CALL to the procedure if the procedure had a return code. Otherwise, this will be NULL.

- [1] The BEGIN_TIME and END_TIME are based on the database server's time zone. The user is responsible for making adjustments for daylight savings time (DST).

ADMIN_TASK_UPDATE procedure - Update an existing task

The ADMIN_TASK_UPDATE procedure updates an administrative task, which is any piece of work that can be encapsulated inside a procedure.

Syntax



The schema is SYSPROC.

Procedure parameters

name
An input argument of type VARCHAR(128) that specifies the name of an existing task. This argument cannot be NULL.

begin_timestamp
An input argument of type TIMESTAMP that specifies the earliest time a task can begin execution. The value of this argument cannot be in the past, and it cannot be later than *end_timestamp*.

When task execution begins depends on how this parameter and the *schedule* parameter are defined:

- If the *begin_timestamp* argument is not NULL:
  - If the *schedule* argument is NULL, the task execution begins at *begin_timestamp*.
  - If the *schedule* argument is not NULL, the task execution begins at the next scheduled time at or after *begin_timestamp*.
- If the *begin_timestamp* argument is NULL:
  - If the *schedule* argument is NULL, the task execution begins immediately.
  - If the *schedule* argument is not NULL, the task execution begins at the next scheduled time.

end_timestamp
An input argument of type TIMESTAMP that specifies the last time that a task can begin execution. The value of this argument cannot be in the past, and it cannot be earlier than *begin_timestamp*. If the argument is NULL, the task can continue to execute as scheduled indefinitely.

An executing task will not be interrupted at its *end_timestamp*.

max_invocations
An input argument of type INTEGER that specifies the maximum number of executions allowed for the task. If the argument is NULL, there is no limit to the number of times the task can execute. If the argument is 0, the task will not execute.

This value applies to the schedule if *schedule* is not NULL.

If both *end_timestamp* and *max_invocations* are specified, *end_timestamp* takes precedence. That is, if the *end_timestamp* timestamp is reached, even though the number of task executions so far has not reached the value of *max_invocations*, the task will not be executed again.

schedule
An input argument of type VARCHAR(1024) that specifies a task execution schedule at fixed points in time. If the argument is NULL, the task is not scheduled at fixed points in time.

The *schedule* string must be specified using the UNIX cron format.

Multiple schedules are not supported.

options
An input argument of type VARCHAR(512). This argument must be NULL.

remarks
An input argument of type VARCHAR(254) that specifies a description of the task. This is an optional argument that can be set to NULL.

Authorization

EXECUTE privilege on the ADMIN_TASK_UPDATE procedure. Unless the database was created with the **RESTRICTIVE** option, EXECUTE privilege is granted to PUBLIC by default.

Although the statement authorization ID might allow the procedure to be executed, a task cannot be updated unless the current session authorization ID matches the session authorization ID that was recorded when the task was created. Users with SYSADM, SYSCTRL, SYSMAINT, or DBADM can update any existing task. Attempting to update a task that was added by a different user returns SQL0551N.

Usage notes

If invalid arguments are passed into the procedure, SQL0171N will be returned. The tokens of the message will indicate which argument is invalid and the name of the procedure.

Changes to the task do not take effect until the unit of work is committed and the scheduler has fetched the updated task definition. Leaving the unit of work uncommitted may delay or prevent the execution of the existing task.

The scheduler checks for updated tasks every 5 minutes. To ensure the task executes as expected, the earliest begin time, as defined by the *begin_timestamp*, *end_timestamp* and *schedule* parameters, should be at least 5 minutes after the unit of work commits.

The database must be active on all database partitions to ensure the task can be executed by the scheduler.

The *begin_timestamp*, *end_timestamp*, and *schedule* are based on the database server's time zone. Special attention is required when scheduling a task during the transition period of daylight savings time (DST). If the task is scheduled to run 2:01 AM and it is the time of year when the time springs forward, the task will not run as the clock skips from 2:00 AM to 3:00 AM. On the other hand, when the time falls back an hour, tasks that were originally scheduled between 2:00 AM and 3:00 AM will execute twice. The user is responsible for making adjustments for daylight savings time to ensure their desired behavior.

When a task is updated, the task's internal invocation counter is reset. To illustrate, consider a recurring task with a *max_invocations* value of `10`. If the task executes 3 times, there are 3 corresponding status records in the ADMIN_TASK_STATUS output. The entries have INVOCATION values of 1, 2, and 3, respectively. Now assume the task creator updates the task. This update will reset the internal invocation counter. The original status records remain in intact. Over time, new status records will be created with INVOCATION values of 1, 2, 3, etc. The BEGIN_TIME can be used to distinguish between the original and updated task execution.

Chapter 6. Audit routines and procedures

AUDIT_ARCHIVE procedure and table function - Archive audit log file

The AUDIT_ARCHIVE procedure and table function both archive the audit log file for the connected database.

Syntax

The schema is SYSPROC.

The syntax is the same for both the procedure and table function.

Procedure and table function parameters

directory
An input argument of type VARCHAR(1024) that specifies the directory where the archived audit file(s) will be written. The directory must exist on the server and the instance owner must be able to create files in that directory. If the argument is null or an empty string, the default directory is used.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, NULL or -2 for an aggregate of all database partitions.

Authorization

Execute privilege on the AUDIT_ARCHIVE procedure or table function.

Examples

Example 1: Archive the audit log(s) for all database partitions to the default directory using the procedure.

```
CALL SYSPROC.AUDIT_ARCHIVE(NULL, NULL)
```

Example 2: Archive the audit log(s) for all database partitions to the default directory using the table function.

```
SELECT * FROM TABLE(SYSPROC.AUDIT_ARCHIVE('', -2)) AS T1
```

Information returned

Table 81. Information returned by the AUDIT_ARCHIVE procedure and table function

Column name	Data type	Description
DBPARTITIONNUM	INTEGER	Partition number of archived file.
PATH	VARCHAR(1024)	Directory location of archived file.
FILE	VARCHAR(1024)	Name of the archived file.
SQLCODE	INTEGER	The SQLCODE received while attempting to archive file.

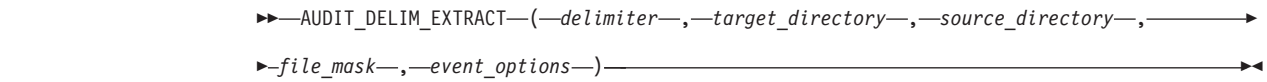
Table 81. Information returned by the AUDIT_ARCHIVE procedure and table function (continued)

Column name	Data type	Description
SQLSTATE	VARCHAR(5)	The SQLSTATE received while attempting archive file. If SQLSTATE is NULL, the value is zero.
SQLERRMC	VARCHAR(70) FOR BIT DATA	The sqlerrmc received while attempting archive file. If SQLSTATE is NULL, the value is zero.

AUDIT_DELIM_EXTRACT - performs extract to delimited file

The AUDIT_DELIM_EXTRACT stored procedure performs an extract to a delimited file on archived audit files of the connected database. Specifically, to those archived audit files that have filenames that match the specified mask pattern.

Syntax



The schema is SYSPROC.

Procedure parameters

delimiter
An optional input argument of type VARCHAR(1) that specifies the character delimiter to be used in the delimited files. If the argument is null or an empty string, a double quote will be used as the delimiter.

target_directory
An optional input argument of type VARCHAR(1024) that specifies the directory where the delimited files will be stored. If the argument is null or an empty string, same directory as the *source_directory* will be used

source_directory
An optional input argument of type VARCHAR(1024) that specifies the directory where the archived audit log files are stored. If the argument is null or an empty string, the audit default will be used.

file_mask
An optional input argument of type VARCHAR(1024) is a mask for which files to extract. If the argument is null or an empty string, it will extract from all audit log files in the source directory.

event_options
An optional input argument of type VARCHAR(1024) that specifies the string defines which events to extract. This matches the same string in the db2audit utility. If the argument is null or an empty string, it will extract all events.

Authorization

Execute privilege on the AUDIT_DELIM_EXTRACT function.

Examples

Note: Audit log files contain a timestamp as part of their naming convention.

*Example 1:*Performs a delimited extract on all audit log files archived on June 18th, 2007 in the default archive directory. This example is extracting just execute events, using a double quote (") character delimiter, and creating or appending the resulting extract files (<category>.del) in the $HOME/audit_delim_extract directory.

```
CALL SYSPROC.AUDIT_DELIM_EXTRACT(NULL, '$HOME/AUDIT_DELIM_EXTRACT', NULL,
'%20070618%', 'CATEGORIES EXECUTE STATUS BOTH')
```

AUDIT_LIST_LOGS table function - Lists archived audit log files

The AUDIT_LIST_LOGS table function lists the archived audit log files for a database which are present in the specified directory.

Syntax

The schema is SYSPROC.

Procedure parameters

directory
An optional input argument of type VARCHAR(1024) that specifies the directory where the archived audit file(s) will be written. The directory must exist on the server and the instance owner must be able to create files in that directory. If the argument is null or an empty string, then the search default directory is used.

Authorization

EXECUTE privilege on AUDIT_LIST_LOGS table function.

Examples

Example 1: Lists all archived audit logs in the default audit archive directory:

```
SELECT * FROM TABLE(SYSPROC.AUDIT_LIST_LOGS('')) AS T1
```

Note: This only lists the logs in the directory for database on which the query is run. Archived files have the format db2audit.db.<dbname>.log.<timestamp>

Information Returned

Table 82. The information returned for AUDIT_LIST_LOGS

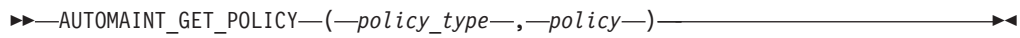
Column Name	Data Type	Description
PATH	VARCHAR(1024)	Path location of the archived file.
FILE	VARCHAR(1024)	Filename of the archived file.
SIZE	BIGINT	File size of the archived file.

Chapter 7. Automatic maintenance routines

AUTOMAINT_GET_POLICY procedure - retrieve automatic maintenance policy

The AUTOMAINT_GET_POLICY system stored procedure retrieves the automatic maintenance configuration for the database. This procedure takes two parameters: the type of automatic maintenance about which to collect information; and a pointer to a BLOB in which to return the configuration information. The configuration information is returned in XML format.

Syntax



The schema is SYSPROC.

Procedure parameters

policy_type
An input argument of type VARCHAR(128) that specifies the type of automatic maintenance policy to retrieve. The value can be one of the following:

AUTO_BACKUP
automatic backup

AUTO_REORG
automatic table and index reorganization

AUTO_RUNSTATS
automatic table runstats operations

MAINTENANCE_WINDOW
maintenance window

policy
An output argument of type BLOB(2M) that specifies the automatic maintenance settings for the given policy type, in XML format.

Authorization

EXECUTE privilege on the AUTOMAINT_GET_POLICY procedure.

Example

Here is an example of a call to the AUTOMAINT_GET_POLICY procedure from within embedded SQL C source code.

- A BLOB variable is declared for the procedure output parameter.
- The procedure is called, specifying automated backup as the type of automatic maintenance policy, and specifying the BLOB variable as the output parameter in which the procedure will return the backup policy for the currently connected database.

```
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS BLOB(2M) backupPolicy;
EXEC SQL END DECLARE SECTION;

EXEC SQL CALL AUTOMAINT_GET_POLICY( 'AUTO_BACKUP', :backupPolicy );
```

AUTOMAINT_GET_POLICYFILE procedure - retrieve automatic maintenance policy

The AUTOMAINT_GET_POLICYFILE system stored procedure retrieves the automatic maintenance configuration for the database. This procedure takes two parameters: the type of automatic maintenance about which to collect information; and the name of a file in which to return the configuration information. The configuration information is returned in XML format.

Syntax

The schema is SYSPROC.

Procedure parameters

policy_type
An input argument of type VARCHAR(128) that specifies the type of automatic maintenance policy to retrieve. The value can be one of the following:

AUTO_BACKUP
automatic backup

AUTO_REORG
automatic table and index reorganization

AUTO_RUNSTATS
automatic table runstats operations

MAINTENANCE_WINDOW
maintenance window

policy_file_name
An input argument of type VARCHAR(2048) that specifies the name of the file that is created in the `tmp` subdirectory of the DB2 instance directory.

Note: The file name may be prefixed with a path relative to tmp. In that case the directory should exist, should have permission to create/overwrite the file and the correct path separator for the DB2 Server must be used.

For example:

On UNIX if the instance directory is defined as `$HOME/sqllib`. For a policy file named `'policy.xml'`, the file name will be `'$HOME/sqllib/tmp/policy.xml'`

On Windows, the instance directory name can be determined from the values of the **DB2INSTPROF** registry variable and the **DB2INSTANCE** environment variable. For a policy file named `'policy.xml'`, if db2set gives `DB2INSTPROF=C:\DB2PROF` and `%DB2INSTANCE%=db2`, then the file name will be `C:\DB2PROF\db2\tmp\policy.xml`

Authorization

EXECUTE privilege on the AUTOMAINT_GET_POLICYFILE procedure.

Example

To get the current automatic maintenance settings for backup operations:

```
call sysproc.automaint_get_policyfile( 'AUTO_BACKUP', 'AutoBackup.xml' )
```

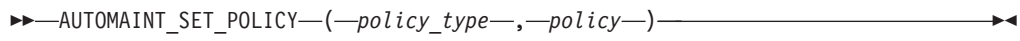
This will create an XML file named `AutoBackup.xml` in the `tmp` subdirectory under the DB2 instance directory.

AUTOMAINT_SET_POLICY procedure - configure automatic maintenance policy

You can use the AUTOMAINT_SET_POLICY system stored procedure to configure automatic maintenance for the database. This procedure takes two parameters: the type of automatic maintenance to configure; and a BLOB containing XML that specifies the configuration.

To enable the reclamation of extents during the automatic reorganization operations on multidimensional clustering (MDC) tables, you need to specify the "reclaimExtentsSizeForMDCTables" attribute to the ReorgOptions element, along with a threshold value in the XML input files. This threshold value is the minimum size, in kilobytes, of free space inside the table that can be reclaimed. This value must be 0 or larger. For example, if you specify a value of 1024 KB for the threshold, only tables with 1 MB of free space or more are considered for automatic reorganization to reclaim extents.

Syntax



The schema is SYSPROC.

Table function parameters

policy_type
An input argument of type VARCHAR(128) that specifies the type of automatic maintenance policy to configure. The value can be one of the following:

AUTO_BACKUP
automatic backup

AUTO_REORG
automatic table and index reorganization

AUTO_RUNSTATS
automatic table runstats operations

MAINTENANCE_WINDOW
maintenance window

policy
An input argument of type BLOB(2M) that specifies the automatic maintenance policy in XML format.

Authorization

EXECUTE privilege on the SYSPROC.AUTOMAINT_SET_POLICY procedure.

Examples

Example 1: To set the current automatic maintenance settings for runstats operations:

```
CALL SYSPROC.AUTOMAINT_SET_POLICY
  ( 'AUTO_RUNSTATS',
    BLOB(' <?xml version=\"1.0\" encoding=\"UTF-8\"?>
      <DB2AutoRunstatsPolicy
          xmlns=\"http://www.ibm.com/xmlns/prod/db2/autonomic/config\">
        <RunstatsTableScope><FilterCondition/></RunstatsTableScope>
      </DB2AutoRunstatsPolicy>')
  )
```

This will replace the current automatic statistics collection configuration with the new configuration contained in the XML document that is passed as the second parameter to the procedure."

Example 2: The automatic reorganization feature of DB2 can use the new "RECLAIM EXTENTS ONLY" option to reorganize multi dimensional clustering (MDC) tables. To enable this feature, set the "reclaimExtentsSizeForMDCTables" value in the AUTO_REORG policy:

```
CALL SYSPROC.AUTOMAINT_SET_POLICY
  ('AUTO_REORG',
  BLOB(' <?xml version=\"1.0\" encoding=\"UTF-8\"?>
    <DB2AutoReorgPolicy
        xmlns=\"http://www.ibm.com/xmlns/prod/db2/autonomic/config\">
      <ReorgOptions dictionaryOption="Keep" indexReorgMode="Online"
          useSystemTempTableSpace="false" reclaimExtentsSizeForMDCTables ="1024" >
      <ReorgTableScope>
        <FilterClause>TABSCHEMA NOT LIKE 'EMP%'</FilterClause>
      </ReorgTableScope>
    </DB2AutoReorgPolicy>')
  )
```

There are sample XML input files located in the `SQLLIB/samples/automaintcfg` directory that you can modify to suit your requirements and then pass the XML content in through the BLOB() scalar function as in the example.

AUTOMAINT_SET_POLICYFILE procedure - configure automatic maintenance policy

You can use the AUTOMAINT_SET_POLICYFILE system stored procedure to configure automatic maintenance for the database. This procedure takes two parameters: the type of automatic maintenance to configure; and the name of an XML document that specifies the configuration.

This procedure return the SQL success or SQL error code.

Syntax

The schema is SYSPROC.

Table function parameters

policy_type
An input argument of type VARCHAR(128) that specifies the type of automatic maintenance policy to configure. The value can be one of the following:

AUTO_BACKUP
automatic backup

AUTO_REORG
automatic table and index reorganization

AUTO_RUNSTATS
automatic table runstats operations

MAINTENANCE_WINDOW
maintenance window

policy_file_name
An input argument of type VARCHAR(2048) that specifies the name of the file that is available in the `tmp` subdirectory of the DB2 instance directory.

Note: When the file name is specified with a relative path, the correct path separator for the DB2 Server must be used and the directory and file should exist with read permission.

For example:

On UNIX if the instance directory is defined as `$HOME/sqllib`. For a policy file named `'automaint/policy.xml'`, the file name will be `'$HOME/sqllib/tmp/automaint/policy.xml'`

On Windows, the instance directory name can be determined from the values of the **DB2INSTPROF** registry variable and the **DB2INSTANCE** environment variable. For a policy file named `'automaint\policy.xml'`, if db2set gives `DB2INSTPROF=C:\DB2PROF` and `%DB2INSTANCE%=db2`, then the file name will be `C:\DB2PROF\db2\tmp\automaint\policy.xml`

Authorization

EXECUTE privilege on the SYSPROC.AUTOMAINT_SET_POLICYFILE procedure.

Example

To modify the current automatic maintenance settings for automatic backup:

```
call sysproc.automaint_set_policyfile( 'AUTO_BACKUP', 'AutoBackup.xml' )
```

This will replace the current automatic backup configuration settings with the new configuration contained in the `AutoBackup.xml` file located in the `tmp` directory under the DB2 instance directory.

There are sample XML input files located in the `SQLLIB/samples/automaintcfg` directory which can be used as reference to create policy xml files.

Chapter 8. Common SQL API procedures

The common SQL API provides a collection of common-signature and signature-stable stored procedures that are portable across IBM data servers. You can use these stored procedures to create applications that perform a variety of common administrative functions, such as getting and setting configuration parameters, and getting system information.

The stored procedures provide syntactically identical XML parameters and error handling across all data servers to ensure data server version independence. Signature-stability and commonality are achieved by using simple XML documents (with a common DTD) as parameters. Version, platform, and technology differences are expressed through different key value pairs in hierarchical property lists.

Common input and output parameters

The common SQL API stored procedures share a set of input and output parameters.

The following table provides a brief description of these parameters. For more detailed information, see the reference topics about the common SQL API stored procedures.

Table 83. Common SQL API shared input and output parameters

Parameter	Description
major_version	Indicates the document type major version that the caller supports for the XML documents passed as parameters in the procedure.
minor_version	Indicates the document type minor version that the caller supports for the XML documents passed as parameters in the procedure. The parameters *major_version* and *minor_version* are used together to ensure that the caller does not use an XML input document of an incorrect version. The procedure processes all XML documents in the specified *major_version* and *minor_version*, or returns an error if a version is not valid. This design supports extensibility in future releases because newer document type versions can be added without affecting existing applications.
requested_locale	Specifies the locale to use to return translated content in the XML documents returned in the *xml_output* and *xml_message* parameters. Only values are translated, not key names.
xml_input	Specifies an XML input document that contains input values for the procedure.

Table 83. Common SQL API shared input and output parameters (continued)

Parameter	Description
xml_filter	Specifies a valid XPath query string that is used to retrieve a single value from an output parameter document.
xml_output	Returns a complete XML output document encoded in UTF-8. Depending on the procedure that is being called, this document might contain configuration parameters and their values, system information, or message text. When the procedure operates in *complete mode*, this parameter returns an XML document that you can modify and pass back to the procedure as the *xml_input* parameter. This approach provides a programmatic way to create valid XML input documents.
xml_message	Returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about a SQL warning condition.

Versioning of XML documents

To support extensibility in future releases, the common SQL API stored procedures return XML output documents that include version information.

Whenever the structure of an XML output document changes (for example, when an element is added or removed), the version levels are incremented. Therefore, a procedure might support several versions of the XML output document.

Version information in the XML document is expressed as key-value pairs for document type major version and document type minor version. For example, an XML output document might define the following keys and values in a dictionary element:

```
<key>Document Type Name</key><string>Data Server Configuration Output</string>
<key>Document Type Major Version</key><integer>2</integer>
<key>Document Type Minor Version</key><integer>0</integer>
```

When you call the procedure, you specify the major version and minor version of the XML document that you want to return. The contents of the XML output document will vary depending on the values that you specify.

For example, the GET_CONFIG procedure retrieves the database and database manager configuration parameters that are set for a particular instance. When this procedure is called with *major_version* 2 and *minor_version* 0, it returns an XML document that contains configuration parameters grouped into categories. However, when the same procedure is called with *major_version* 1 and *minor_version* 0, it returns an XML document that contains configuration parameters, but they are not grouped into categories.

Likewise, the GET_MESSAGE procedure retrieves the message text and SQLSTATE for a specified SQLCODE. When this procedure is called with *major_version* 2 and *minor_version* 0, it returns an XML document that contains the short text message, long text message, and SQLSTATE for the corresponding SQLCODE. However,

when the same procedure is called with *major_version* 1 and *minor_version* 0, it returns an XML document that contains only the short text message and SQLSTATE. The long text message is not available in version 1 of the document.

To determine the highest supported document versions for a procedure, specify NULL for *major_version*, *minor_version*, and all other input parameters. The procedure returns the highest supported document versions as values in the *major_version* and *minor_version* output parameters, and sets the *xml_output* and *xml_message* output parameters to NULL.

If you specify non-null values for *major_version* and *minor_version*, you must specify supported document versions, or the procedure raises an error (-20457) to indicate that the procedure encountered an unsupported version.

XML input documents can optionally include values for the document type major version and document type minor version. If these values are specified in the XML input document, then the values passed for *major_version* and *minor_version* in the procedure call must exactly match the values that are specified in the XML document, or the procedure raises an error (+20458). This behavior ensures that the caller does not specify an unsupported version of the XML input document.

XML input documents

The XML documents that are passed as input to common SQL API stored procedures share a simple XML format that is based on a common DTD.

The XML input document consists of a set of entries that are common to all stored procedures, and a set of entries that are specific to each stored procedure. The XML input document has the following general structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Message Input</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Document Locale</key><string>en_US</string>
   <key>Complete</key><false/>
   <dict>
      <!-- Document type specific data appears here. -->
   </dict>
</dict>
</plist>
```

Important: XML input documents must be encoded in UTF-8 and contain English characters only.

Complete mode for returning valid XML input documents

You can use *complete mode* to create a valid XML document for any common SQL API stored procedure that accepts input. You can then customize the document and pass it back to the procedure.

To run a procedure in complete mode, specify "true" for the Complete key in the input XML document, and pass the following minimal content:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Complete</key><true/>
</dict>
</plist>
```

Any XML elements that are not required are ignored and are not returned in the output document.

When you run the procedure, a complete XML input document is returned in the *xml_output* parameter of the stored procedure. The returned XML document includes a document type and a section for all possible required parameters and optional parameters. The returned XML document also includes other entries (such as display names, hints, and the document locale) that are not required, but are typically needed when rendering the document in a client application.

After rendering the XML document and modifying it in a platform-independent way, you can run the same stored procedure and pass in the modified XML document as input.

XML output documents

The XML documents that are returned as output from common SQL API stored procedures share a common set of entries.

At a minimum, XML documents returned in the *xml_output* parameter include the following mandatory, key value pairs:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key>
   <string>Data Server Configuration Output</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Data Server Product Name</key><string>DSN</string>
   <key>Data Server Product Version</key><string>8.1.0.356</string>
   <key>Data Server Major Version</key><integer>8</integer>
   <key>Data Server Minor Version</key><integer>1</integer>
   <key>Data Server Platform</key><string>z/OS</string>
   <key>Document Locale</key><string>en_US</string>

   <!-- Document type specific data appears here. -->
</dict>
</plist>
```

Entries in the XML output document might be grouped using nested dictionaries. Each entry in the XML output document describes a single piece of information. The entry consists of the value, the display name, and a hint. Optionally, a display unit might be provided. Display name, hint, and display unit are language-sensitive and will be translated to the language specified in the value for the *requested_locale* parameter (or the default if the requested locale is not yet supported). In general, an entry has a structure similar to the following example:

```
<key>Real Storage Size</key>
<dict>
     <key>Display Name</key><string>Real Storage Size</string>
     <key>Value</key><integer>2048</integer>
     <key>Display Unit</key><string>MB</string>
     <key>Hint</key><string>Size of actual real storage online</string>
</dict>
```

IBM data servers have a common parameter document that includes some keywords that are applicable to all data servers, and others that are data server specific. Whenever a data server adds or removes a new keyword, the version number (for all data servers) is incremented. Depending on the change, the major version number might be increased and the minor version number set to 0 (zero), or only the minor version number might be incremented.

XML output documents are generated in UTF-8 and contain English characters only.

XPath expressions for filtering output

You can use an XPath expression to filter the XML output returned by a common SQL API stored procedure.

To filter the output, specify a valid XPath query string in the *xml_filter* parameter of the procedure. The following restrictions apply to the XPath expression that you specify:

- The XPath expression must reference a single value.
- The XPath expression must always be absolute from the root node. For example, the following path expressions are allowed: `/`, *`nodename`*, `.`, and `..`. The following expressions are not allowed: `//` and `@`
- The only predicates allowed are [*`path`*`='`*`value`*`'`] and [*`n`*].
- The only axis allowed is `following-sibling`.
- The XPath expression must end with one of the following, and, if necessary, be appended with the predicate [1]: `following-sibling::string`, `following-sibling:: data`, `following-sibling::date`, `following-sibling::real`, or `following-sibling::integer`.
- Unless the axis is found at the end of the XPath expression, it must be followed by a `::dict`, `::string`, `::data`, `::date`, `::real`, or `::integer`, and if necessary, be appended with the predicate [1].
- The only supported XPath operator is `=`.
- The XPath expression cannot contain a function, namespace, processing instruction, or comment.

Tip: If the stored procedure operates in *complete mode*, do not apply filtering, or a SQLCODE (+20458) is raised.

For better control over processing the XML document returned in the *xml_output* parameter, you can use the XMLPARSE function available with DB2 pureXML®.

Example

The following XPath expression selects the value for the Data Server Product Version key from an XML output document:

```
/plist/dict/key[.='Data Server Product Version']following-sibling::string[1]
```

The procedure returns the string `8.1.0.356` in the *xml_output* parameter. Therefore, the procedure call returns a single value rather than an XML document.

XML message documents

When a common SQL API stored procedure encounters an internal processing error or invalid parameter, the data server returns a SQLCODE and the corresponding SQL message to the caller. When this occurs, the procedure returns an XML message document in the *xml_message* parameter that contains more detailed information about the warning situation.

The XML message document has the following general structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Message</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Data Server Product Name</key><string>QDB2/AIX64</string>
   <key>Data Server Product Version</key><string>9.5.0.3</string>
   <key>Data Server Major Version</key><integer>9</integer>
   <key>Data Server Minor Version</key><integer>5</integer>
   <key>Data Server Platform</key><string>AIX 64BIT</string>
   <key>Document Locale</key><string>en_US</string>
   <key>Short Message Text</key>
   <dict>
      <key>Value</key><string>
      <!-- Additional description of warning appears here. --></string>
      <key>Hint</key><string></string>
   </dict>
</dict>
</plist>
```

XML message documents are generated in UTF-8 and contain English characters only.

Example

In the following example, a call to the GET_MESSAGE procedure results in an SQL warning:

```
db2 "CALL SYSPROC.GET_MESSAGE(NULL,NULL,'en_US',NULL,NULL,?,?)"

SQL20458W The procedure "SYSPROC.GET_MESSAGE" has encountered
an internal parameter processing error in parameter "3".
The value for parameter "7" contains further information about
the error. SQLSTATE=01H54
```

The XML document that is returned in parameter 7 (*xml_message*) contains the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Message</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Data Server Product Name</key><string>QDB2/AIX64</string>
   <key>Data Server Product Version</key><string>9.5.0.3</string>
   <key>Data Server Major Version</key><integer>9</integer>
   <key>Data Server Minor Version</key><integer>5</integer>
   <key>Data Server Platform</key><string>AIX 64BIT</string>
   <key>Document Locale</key><string>en_US</string>
   <key>Short Message Text</key>
   <dict>
      <key>Value</key><string>If parameters 1 and 2 are set to NULL, all
      other input parameters must be set to NULL as well, but the value
      of parameter "3" is not NULL. </string>
```

```
      <key>Hint</key><string></string>
   </dict>
</dict>
</plist>
```

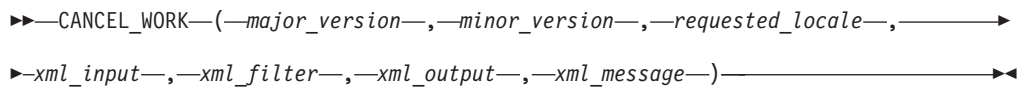
The value for the Short Message Text key provides additional information about the warning.

CANCEL_WORK procedure - Cancel work

The CANCEL_WORK stored procedure cancels either a specific activity (for example, a SQL statement), or all activity for a connected application.

To cancel a specific activity, you pass in the application handle, unit of work ID, and activity ID for the activity that you want to cancel. To cancel all activity for a connected application, you pass in the application handle. Any changes associated with the cancelled activity are rolled back.

Syntax



The schema is SYSPROC.

Procedure parameters

major_version
An input and output argument of type INTEGER that indicates the major document version. On input, this argument indicates the major document version that the caller supports for the XML documents passed as parameters in the procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error (+20458) if the version is not valid. On output, this parameter specifies the highest major document version that is supported by the procedure. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

Supported versions: 1

minor_version
An input and output argument of type INTEGER that indicates the minor document version. On input, this argument specifies the minor document version that the caller supports for the XML documents passed as parameters for this procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error if the version is not valid. On output, this parameter indicates the highest minor document version that is supported for the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

Supported versions: 0

requested_locale
An input argument of type VARCHAR(33) that specifies a locale. If the

specified language is supported on the server, translated content is returned in the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names and values are not translated. The only translated portion of the XML output and XML message documents are the text for hint, display name, and display unit of each entry. The caller should always compare the requested language to the language that is used in the XML output document (see the document locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input
An input argument of type BLOB(32MB) that specifies an XML input document (encoded in UTF-8) that contains input values for the procedure.

For this procedure, the XML input document must specify an application handle. If you want to cancel a specific activity, the XML input document must also specify optional parameters that identify a unit of work ID and an activity ID. A complete XML input document for this stored procedure looks something like the following document:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Cancel Work Input</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Required Parameters</key>
   <dict>
      <key>Application Handle</key>
      <dict>
         <key>Display name</key><string>Application Handle</string>
         <key>Value</key><integer>10</integer>
         <key>Hint</key>
         <string>
         Numeric value equivalent to the application handle to be cancelled
         </string>
      </dict>
   </dict>
   <key>Optional Parameters</key>
   <dict>
      <key>Unit Of Work Id</key>
      <dict>
         <key>Display Name</key><string>Unit Of Work Id</string>
         <key>Value</key><integer>20</integer>
         <key>Hint</key>
         <string>
         Numeric value that specifies the unit of work id of the activity
         that is to be cancelled
         </string>
      </dict>
      <key>Activity Id</key>
      <dict>
         <key>Display Name</key><string>Activity Id</string>
         <key>Value</key><integer>10</integer>
         <key>Hint</key>
         <string>
         Numeric value equivalent to the activity id to be cancelled
         </string>
      </dict>
   </dict>
</dict>
</plist>
```

If you specify the application handle of the application where the stored procedure is running, the procedure returns a warning (SQL20458).

xml_filter
An input argument of type BLOB(4K) that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see the topic that describes XPath filtering.

The following example selects the value for the Data Server Product Version from the XML output document: `/plist/dict/key[.='Data Server Product Version']/following-sibling::string`. If the key is not followed by the specified sibling, an error is returned.

xml_output
An output parameter of type BLOB(32MB) that returns a complete XML output document in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

The XML output is determined by the values that you specify for *major_version* and *minor_version*:

Major version	**Minor version**	*xml_output* **value**
NULL	NULL	NULL
1	0	The status of the activity that the procedure attempted to cancel.

When the procedure operates in *complete mode*, this parameter returns an XML document that you can modify and pass back to the procedure as the *xml_input* parameter. This approach provides a programmatic way to create valid XML input documents. For more information, see the topic about complete mode.

xml_message
An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about a SQL warning condition. This document is returned when a call to the procedure results in a SQL warning, and the warning message indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

Authorization

- SYSADM or DBADM authority
- EXECUTE privilege on the CANCEL_WORK procedure

Example

Example 1: Return the highest supported version of the procedure.

```
db2 "call sysproc.cancel_work(null,null,null,null,null,?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
```

```
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : -

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

Example 2: Cancel a specific activity.

```
db2 "call sysproc.cancel_work(1,0,'en_US',blob(
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Cancel Work Input</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Required Parameters</key>
   <dict>
      <key>Application Handle</key>
      <dict>
          <key>Display name</key><string>Application Handle</string>
          <key>Value</key><integer>1</integer>
          <key>Hint</key>
          <string>
          Numeric value equivalent to the application handle to be cancelled
          </string>
      </dict>
   </dict>
   <key>Optional Parameters</key>
   <dict>
      <key>Unit Of Work Id</key>
      <dict>
          <key>Display Name</key><string>Unit Of Work Id</string>
          <key>Value</key><integer>2</integer>
          <key>Hint</key>
          <string>
          Numeric value that specifies the unit of work id of the activity
          that is to be cancelled
          </string>
      </dict>
      <key>Activity Id</key>
      <dict>
         <key>Display Name</key><string>Activity Id</string>
         <key>Value</key><integer>3</integer>
         <key>Hint</key>
         <string>
         Numeric value equivalent to the activity id to be cancelled
         </string>
      </dict>
   </dict>
</dict>
</plist> ) ,null,?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0
```

```
Parameter Name : XML_OUTPUT
Parameter Value : x'3C3F78...'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

If the CANCEL_WORK procedure is able to cancel the activity, the XML output document contains the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict><key>Document Type Name</key><string>Data Server Cancel Work Output</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Data Server Product Name</key><string>QDB2/AIX64</string>
   <key>Data Server Product Version</key><string>9.7.0.0</string>
   <key>Data Server Major Version</key><integer>9</integer>
   <key>Data Server Minor Version</key><integer>7</integer>
   <key>Data Server Platform</key><string>AIX 64BIT</string>
   <key>Document Locale</key><string>en_US</string>
   <key>Successful Cancel Work Message</key>
   <dict>
      <key>Display Name</key><string>Successful Cancel Work Message</string>
      <key>Value</key><string>The activity has been cancelled successfully</string>
      <key>Hint</key><string></string>
   </dict>
</dict>
</plist>
```

Example 2: Cancel the application.

```
db2 "call sysproc.cancel_work(1,0,'en_US,blob(
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Cancel Work Input</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Required Parameters</key>
   <dict>
      <key>Application Handle</key>
      <dict>
          <key>Display name</key><string>Application Handle</string>
          <key>Value</key><integer>101</integer>
          <key>Hint</key>
          <string>
          Numeric value equivalent to the application handle to be cancelled
          </string>
      </dict>
   </dict>
</dict>
</plist> ),null,?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'3C3F78...'
```

```
Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

If the CANCEL_WORK procedure is able to cancel the application, the XML output document contains the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Cancel Work Output</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Data Server Product Name</key><string>QDB2/AIX64</string>
   <key>Data Server Product Version</key><string>9.7.0.0</string>
   <key>Data Server Major Version</key><integer>9</integer>
   <key>Data Server Minor Version</key><integer>7</integer>
   <key>Data Server Platform</key><string>AIX 64BIT</string>
   <key>Document Locale</key><string>en_US</string>
   <key>Successful Cancel Work Message</key>
   <dict>
      <key>Display Name</key><string>Successful Cancel Work Message</string>
      <key>Value</key>
      <string>The application has been cancelled successfully</string>
      <key>Hint</key><string></string>
   </dict>
</dict>
</plist>
```

Example 3: Specify a filter to return the value of a successful cancel work message.

```
db2 "call sysproc.cancel_work(1,0,'en_US,blob(
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Cancel Work Input</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Required Parameters</key>
   <dict>
      <key>Application Handle</key>
      <dict>
          <key>Display name</key><string>Application Handle</string>
          <key>Value</key><integer>101</integer>
          <key>Hint</key>
          <string>
          Numeric value equivalent to the application handle to be cancelled
          </string>
      </dict>
   </dict>
</dict>
</plist> ),blob('/plist/dict/key[.="Successful Cancel Work Message"]
/following-sibling::dict[1]/key[.="Value"]
/following-sibling::string[1]'),?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
```

```
Parameter Value : x'3C3F78...'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

The following value is returned for *xml_output*:

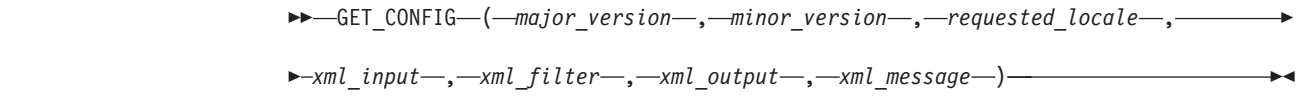
```
"The application has been cancelled successfully"
```

GET_CONFIG procedure - Get configuration data

The GET_CONFIG stored procedure retrieves the database configuration, database manager configuration, and registry variables that are set for a particular instance.

In a partitioned database environment, this procedure retrieves database configuration and registry variable settings from all partitions.

Syntax



The schema is SYSPROC.

Procedure parameters

major_version
An input and output argument of type INTEGER that indicates the major document version. On input, this argument indicates the major document version that the caller supports for the XML documents passed as parameters in the procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error (+20458) if the version is not valid. On output, this parameter specifies the highest major document version that is supported by the procedure. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

Supported versions: 1 and 2

minor_version
An input and output argument of type INTEGER that indicates the minor document version. On input, this argument specifies the minor document version that the caller supports for the XML documents passed as parameters for this procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error if the version is not valid. On output, this parameter indicates the highest minor document version that is supported for the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

Supported versions: 0

requested_locale
An input argument of type VARCHAR(33) that specifies a locale. If the

specified language is supported on the server, translated content is returned in the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names and values are not translated. The only translated portion of the XML output and XML message documents are the text for hint, display name, and display unit of each entry. The caller should always compare the requested language to the language that is used in the XML output document (see the document locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input
Currently, this procedure accepts no input. You must specify NULL for this parameter, or an error (+20458) is raised to indicate that the input is not valid.

xml_filter
An input argument of type BLOB(4K) that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see the topic that describes XPath filtering.

The following example selects the value for the data server product version from the XML output document: `/plist/dict/key[.='Data Server Product Version']/following-sibling::string`. If the key is not followed by the specified sibling, an error is returned.

xml_output
An output parameter of type BLOB(32MB) that returns a complete XML output document in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

The XML output is determined by the values that you specify for *major_version* and *minor_version*:

Major version	**Minor version**	*xml_output* **value**
NULL	NULL	NULL
1	0	Database manager and database configuration parameters and registry variables, including their values.
2	0	Database manager and database configuration parameters grouped into categories. For each parameter, indicates whether the parameter can be updated. Also returns registry variables and the values set for the instance.

When the procedure operates in *complete mode*, this parameter returns an XML document that you can modify and pass back to the procedure as the *xml_input* parameter. This approach provides a programmatic way to create valid XML input documents. For more information, see the topic about complete mode.

xml_message
An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about a SQL warning condition. This document is returned when a call to the procedure results in a SQL warning, and the warning message indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

Authorization

- SYSADM or DBADM authority
- EXECUTE privilege on the GET_CONFIG procedure

Example

Example 1: Return the latest version of the procedure.

```
db2 "call sysproc.get_config(null,null,null,null,null,?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 2

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : -

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

Example 2: Return database and database manager configuration parameters, grouped into categories.

```
db2 "call sysproc.get_config(2,0,'en_US',null, null, ?,?)"
```

The following is an example of output from this query:

```
 Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 2

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'3C3F78.......'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

The XML output document contains the following content:

```
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Configuration Output</string>
```

```
<key>Document Type Major Version</key><integer>2</integer>
<key>Document Type Minor Version</key><integer>0</integer>
<key>Data Server Product Name</key><string>QDB2/AIX64</string>
<key>Data Server Product Version</key><string>9.7.0.0</string>
<key>Data Server Major Version</key><integer>9</integer>
<key>Data Server Minor Version</key><integer>7</integer>
<key>Data Server Platform</key><string>AIX 64BIT</string>
<key>Document Locale</key><string>en_US</string>
<key>Database Manager Configuration Parameter Settings</key>
<dict>
   <key>Display Name</key>
   <string>Database Manager Configuration Parameter Settings</string>
   <key>Application</key>
   <dict>
      <key>Display Name</key><string>Application</string>
      <key>agentpri</key>
      <dict>
         <key>Display Name</key><string>agentpri</string>
         <key>Parameter Value</key>
         <dict>
            <key>Display Name</key><string>Parameter Value</string>
            <key>Value</key><string>-1</string>
            <key>Updatable</key><string>No</string>
            <key>Hint</key><string></string>
         </dict>
         <key>Value Flags</key>
         <dict>
            <key>Display Name</key><string>Value Flags</string>
            <key>Value</key><string>NONE</string>
            <key>Updatable</key><string>No</string>
            <key>Hint</key><string></string>
         </dict>
         <key>Deferred Value</key>
         <dict>
            <key>Display Name</key><string>Deferred Value</string>
            <key>Value</key><string>-1</string>
            <key>Updatable</key><string>Yes</string>
            <key>Hint</key><string></string>
         </dict>
         <key>Deferred Value Flags</key>
         <dict>
            <key>Display Name</key><string>Deferred Value Flags</string>
            <key>Value</key><string>INTEGER</string>
            <key>Updatable</key><string>Yes</string>
            <key>Hint</key><string></string>
         </dict>
         <key>Data Type</key>
         <dict>
            <key>Display Name</key><string>Data Type</string>
            <key>Value</key><string>NONE</string>
            <key>Hint</key><string></string>
         </dict>
         <key>Hint</key>
         <string>
         Specifies the priority given to an agent and other database manager
         instance processes and threads by the operating system scheduler.
         Consider rebinding applications after changing this parameter.
         </string>
      </dict>
      <key>Hint</key><string></string>
   </dict>
</dict>
<key>Administration</key>
.
.
.
<key>Communication</key>
```

```
.
.
.
<key>Diagnostics</key>
.
.
.
<key>Environment</key>
.
.
.
<key>Miscellaneous</key>
.
.
.
<key>Monitor</key>
.
.
.
<key>Parallel</key>
.
.
.
<key>Performance</key>
.
.
.
</dict>
<key>Database Partition</key>
<dict>
   <key>Display Name</key><string>Database Partition</string>
   <key>0</key>
   <dict>
      <key>Display Name</key><string>0</string>
      <key>Database Configuration Parameter Settings</key>
      <dict>
         <key>Display Name</key>
         <string>Database Configuration Parameter Settings</string>
         <key>Application</key>
         .
         .
         .
         <key>Environment</key>
      <dict>
         <key>Display Name</key><string>Environment</string>
         <key>alt_collate</key>
         <dict>
            <key>Display Name</key><string>alt_collate</string>
            <key>Parameter Value</key>
            <dict>
               <key>Display Name</key><string>Parameter Value</string>
               <key>Value</key><string></string>
               <key>Updatable</key><string>No</string>
               <key>Hint</key><string></string>
            </dict>
            <key>Value Flags</key>
            <dict>
               <key>Display Name</key><string>Value Flags</string>
               <key>Value</key><string>NONE</string>
               <key>Updatable</key><string>No</string>
               <key>Hint</key><string></string>
            </dict>
            <key>Deferred Value</key>
            <dict>
               <key>Display Name</key><string>Deferred Value</string>
               <key>Value</key><string></string>
               <key>Updatable</key><string>Yes</string>
```

```
            <key>Hint</key><string></string>
         </dict>
         <key>Deferred Value Flags</key>
         <dict>
            <key>Display Name</key><string>Deferred Value Flags</string>
            <key>Value</key><string>INTEGER</string>
            <key>Updatable</key><string>Yes</string>
            <key>Hint</key><string></string>
         </dict>
         <key>Data Type</key>
         <dict>
            <key>Display Name</key><string>Data Type</string>
            <key>Value</key><string>NONE</string>
            <key>Hint</key><string></string>
         </dict>
         <key>Hint</key>
         <string>
         Specifies the collating sequence to be used for Unicode tables in a
         non-Unicode database. Until this parameter is set, Unicode tables and
         routines cannot be created in a non-Unicode database. When set, this
         parameter cannot be changed or reset. Default [range] :
         Null [IDENTITY_16BIT].
         </string>
      </dict>
      .
      .
      .
   </dict>
   <key>Logs</key>
   .
   .
   .
   <key>Maintenance</key>
   .
   .
   .
   <key>Performance</key>
   .
   .
   .
   <key>Recovery</key>
   .
   .
   .
   <key>Status</key>
   .
   .
   .
</dict>
<key>Registry Variables Settings</key>
<dict>
   <key>Display Name</key><string>Registry Variables Settings</string>
   <key>DB2CODEPAGE</key>
   <dict>
      <key>Display Name</key><string>DB2CODEPAGE</string>
      <key>Parameter Value</key>
      <dict>
         <key>Display Name</key><string>Parameter Value</string>
         <key>Value</key><string>1208</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Is Aggregate</key>
      <dict>
         <key>Display Name</key><string>Is Aggregate</string>
         <key>Value</key><integer>0</integer>
         <key>Hint</key><string></string>
      </dict>
```

```
            <key>Aggregate Name</key>
            <dict>
               <key>Display Name</key><string>Aggregate Name</string>
               <key>Value</key><string></string>
               <key>Hint</key><string></string>
            </dict>
            <key>Level</key>
            <dict>
               <key>Display Name</key><string>Level</string>
               <key>Value</key><string>I</string>
               <key>Hint</key><string></string>
            </dict>
            <key>Hint</key><string></string>
         </dict>
         .
         .
         .
      </dict>
      <key>Hint</key><string></string>
   </dict>
</dict>
</plist>
```

Example 3: Return database and database manager configuration parameters.

```
db2 "call sysproc.get_config(1,0,'en_US',null, null, ?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'3C3F78.......'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

The XML output document contains content that is similar to example 2, but does not group the configuration parameters into categories.

Example 4: Call the procedure from a function.

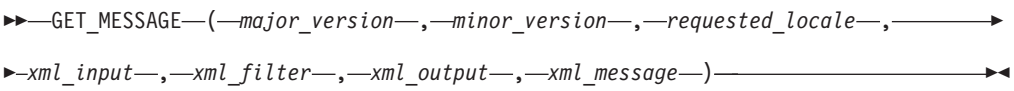
```
EXEC SQL BEGIN DECLARE SECTION;
  sqlint16  getconfigMaj;
  sqlint16  getconfigMin;

  SQL TYPE IS BLOB(2M) xmlOutput;
  SQL TYPE IS BLOB(2K) xmlOutMessage;
EXEC SQL END DECLARE SECTION;
  getconfigMaj = 2;
  getconfigMin = 0;

  EXEC SQL CALL SYSPROC.GET_CONFIG(
                :getconfigMaj,
                :getconfigMin,
                'en_US',
```

```
null,
null,
:xmlOutput,
:xmlOutMessage );
```

GET_MESSAGE procedure - Get message text

The GET_MESSAGE procedure returns the short message text, long message text, and SQLSTATE for an SQLCODE.

Syntax



The schema is SYSPROC.

Procedure parameters

major_version
An input and output argument of type INTEGER that indicates the major document version. On input, this argument indicates the major document version that the caller supports for the XML documents passed as parameters in the procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error (+20458) if the version is not valid. On output, this parameter specifies the highest major document version that is supported by the procedure. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

If the XML document in the *xml_input* parameter specifies a Document Type Major Version key, the value for that key must be equal to the value provided in the *major_version* parameter, or an error (+20458) is raised.

Supported versions: 1 and 2

minor_version
An input and output argument of type INTEGER that indicates the minor document version. On input, this argument specifies the minor document version that the caller supports for the XML documents passed as parameters for this procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error if the version is not valid. On output, this parameter indicates the highest minor document version that is supported for the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

If the XML document in the *xml_input* parameter specifies a Document Type Minor Version key, the value for that key must be equal to the value provided in the *minor_version* parameter, or an error (+20458) is raised.

Supported versions: 0

requested_locale
An input argument of type VARCHAR(33) that specifies a locale. If the specified language is supported on the server, translated content is returned in

the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names and values are not translated. The only translated portion of the XML output and XML message documents are the text for hint, display name, and display unit of each entry. The caller should always compare the requested language to the language that is used in the XML output document (see the document locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input
An input argument of type BLOB(32MB) that specifies an XML input document (encoded in UTF-8) that contains input values for the procedure.

For this procedure, the XML input document contains an SQLCODE and uses the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
    <key>Document Type Name</key><string>Data Server Message Input</string>
    <key>Required Parameters</key>
    <!-- Specify either SQLCODE or message identifier and message tokens
    for the key values below. -->
    <dict>
        <key>SQL Code</key><integer></integer>
        <key>Message Identifier</key><integer></integer>
        <key>Message Tokens</key><array><string>...</string></array>
    </dict>
    <key>Optional Parameters</key>
    <dict>
        <key>Message Token Delimiter<key><string>;</string>
    </key></key></dict>
</dict>
</plist>
```

xml_filter
An input argument of type BLOB(4K) that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see the topic that describes XPath filtering.

The following example selects the value for the SQLSTATE from the XML output document: `/plist/dict/key[.="SQLSTATE"]/following-sibling::dict[1]/key[.="Value"]/following-sibling::string[1]`. If the key is not followed by the specified sibling, an error is returned.

xml_output
An output parameter of type BLOB(32MB) that returns a complete XML output document in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

The XML output is determined by the values that you specify for *major_version* and *minor_version*:

Major version	**Minor version**	*xml_output* **value**
NULL	NULL	NULL

Major version	Minor version	*xml_output* value
1	0	Returns the short text message and SQLSTATE for the corresponding SQLCODE passed in *xml_input*.
2	0	Returns the short text message, long text message and SQLSTATE for the corresponding SQLCODE passed in *xml_input*.

When the procedure operates in *complete mode*, this parameter returns an XML document that you can modify and pass back to the procedure as the *xml_input* parameter. This approach provides a programmatic way to create valid XML input documents. For more information, see the topic about complete mode.

xml_message
An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about a SQL warning condition. This document is returned when a call to the procedure results in a SQL warning, and the warning message indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

Authorization

- SYSADM or DBADM authority
- EXECUTE privilege on the GET_MESSAGE procedure

Example

Example 1: Return the highest supported version of the procedure.

```
db2 "call sysproc.get_message(null,null,null,null,null,?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 2

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : -

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

Example 2: Run a script called `getmsglong.sql` to return the short text message and long text message for SQL1034.

```
getmsglong.sql:

call sysproc.get_message(2,0, 'en_US', blob('
<?xml version="1.0" encoding="UTF-8"?>
```

```
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server Message Input</string>
  <key>Document Type Major Version</key><integer>2</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Required Parameters</key>
  <dict>
    <key>SQLCODE</key><string>SQL1034</string>
  </dict>
</dict>
</plist>'), null, ? , ?)@
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 2

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'3C3F786D6C20766572.....................'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

The output XML document contains the following content:

```
<plist version="1.0">
<dict>
  <key>Document Type Name</key>
  <string>Data Server Message Output</string>
  <key>Document Type Major Version</key>
  <integer>2</integer>
  <key>Document Type Minor Version</key>
  <integer>0</integer>
  <key>Data Server Product Name</key>
  <string>QDB2/AIX64</string>
  <key>Data Server Product Version</key>
  <string>9.7.0.0</string>
  <key>Data Server Major Version</key>
  <integer>9</integer>
  <key>Data Server Minor Version</key>
  <integer>7</integer>
  <key>Data Server Platform</key>
  <string>AIX 64BIT</string>
  <key>Document Locale</key>
  <string>en_US</string>
  <key>Short Message Text</key>
  <dict>
  <key>Display Name</key><string>Short Message Text</string>
  <key>Value</key>
  <string>
  SQL1034C  The database is damaged.  All applications processing the database
  have been stopped.
  </string>
  <key>Hint</key><string></string>
</dict>
  <key>SQLSTATE</key>
  <dict>
    <key>Display Name</key><string>SQLSTATE</string>
    <key>Value</key><string> 58031</string>
    <key>Hint</key><string></string>
  </dict>
```

```
<key>Long Message Text</key>
<dict>
  <key>Display Name</key><string>Long Message Text</string>
  <key>Value</key>
  <array>
    <string>
    SQL1034C  The database is damaged. All applications
    processing the
    </string>
    <string>      database have been stopped.</string>
    <string></string>
    <string>Explanation: </string>
    <string></string>
    <string>
    Damage has occurred to the database. It cannot be used until it is
    </string>
    <string>
    recovered. All applications connected to the database have been
    </string>
    <string>
    disconnected and all processes running applications on the
    database have
    </string>
    <string>been stopped.</string>
    <string></string>
    <string>The command cannot be processed.</string>
    <string></string>
    <string>User response: </string>
    <string></string>
    <string>
    Issue a RESTART DATABASE command to recover the database. If the RESTART
    </string>
    <string>
    command consistently fails, you may want to restore the database from a
    </string>
    <string>
    backup. In a partitioned database server environment, check the syslog
    </string>
    <string>
    to find out if the RESTART command fails because of node or
    </string>
    <string>
    communication failures before restoring the database from a backup. If
    </string>
    <string>
    so, ensure the database manager is up and running and communication is
    </string>
    <string>
    available among all the nodes, then resubmit the restart command.
    </string>
    <string></string>
    <string>
    If you encountered this error during roll-forward processing, you must
    </string>
    <string>
    restore the database from a backup and perform roll-forward again.
    </string>
    <string></string>
    <string>
    Note that in a partitioned database environment, the RESTART database
    </string>
    <string>
    command is run on a per-node basis. To ensure that the database is
    </string>
    <string>restarted on all nodes, use the command: </string>
    <string></string>
    <string>db2_all db2 restart database</string>
```

```
      <string><database_name></string>
      <string></string>
      <string>
      This command may have to be run several times to ensure that all
      </string>
      <string>in-doubt transactions have been resolved.</string>
      <string></string>
      <string>
      If you are installing the sample database, drop it and install the
      </string>
      <string>sample database again.</string>
      <string></string>
      <string> sqlcode: -1034</string>
      <string></string>
      <string> sqlstate: 58031</string>
      <string></string>
      <string></string>
      <string></string>
      </array>
    <key>Hint</key><string></string>
  </dict>
</dict>
</plist>
```

Example 3: Run a script called `getmsgshort.sql` to return only the short text message for SQL1034.

`getmsgshort.sql`:

```
call sysproc.get_message(1,0,'en_US', blob('
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server Message Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Required Parameters</key>
  <dict>
    <key>SQLCODE</key><string>SQL1034</string>
  </dict>
</dict>
</plist>'), null, ? , ?)@
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 2

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'3C3F786D6C20766572.....................'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0

SQL20460W The procedure "SYSPROC.GET_MESSAGE" supports a higher version, "2",
than the specified version, "1", for parameter "1".
```

The XML output document contains the following content:

```
<plist version="1.0">
<dict><key>Document Type Name</key><string>Data Server Message Output</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Data Server Product Name</key><string>QDB2/AIX64</string>
  <key>Data Server Product Version</key><string>9.7.0.0</string>
  <key>Data Server Major Version</key><integer>9</integer>
  <key>Data Server Minor Version</key><integer>7</integer>
  <key>Data Server Platform</key><string>AIX 64BIT</string>
  <key>Document Locale</key><string>en_US</string>
  <key>Short Message Text</key>
  <dict>
    <key>Display Name</key><string>Short Message Text</string>
    <key>Value</key>
    <string>
    SQL1034C  The database is damaged.  All applications processing the database
    have been stopped.
    </string>
    <key>Hint</key><string></string>
  </dict>
  <key>SQLSTATE</key>
  <dict>
    <key>Display Name</key><string>SQLSTATE</string>
    <key>Value</key><string> 58031</string>
    <key>Hint</key><string></string>
  </dict>
</dict>
</plist>
```

Example 4: Specify a filter to return the SQLSTATE for SQL1034.

```
db2 "call sysproc.get_message(2,0, 'en_US', blob('
<plist version="1.0">
<dict>
   <key>Document Type Name</key>
   <string>Data Server Message Input</string>
   <key>Required Parameters</key>
   <dict>
      <key>SQLCODE</key><string>SQL1034</string>
   </dict>
</dict>
</plist>'),
blob('/plist/dict/key[.="SQLSTATE"]/following-sibling::dict[1]/
key[.="Value"]/following-sibling::string[1]'), ? , ?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 2

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'203538303331'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

The following value is returned for *xml_output*:

```
58031
```

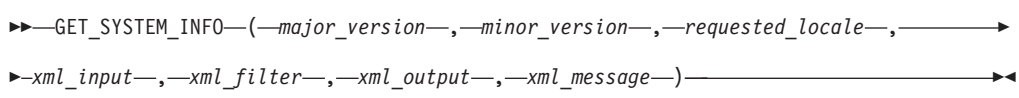
Example 5: Call the procedure from a function.

```
EXEC SQL BEGIN DECLARE SECTION;
  sqlint16  getMsgMaj;
  sqlint16  getMsgMin;

  SQL TYPE IS BLOB(2M) xmlOutput;
  SQL TYPE IS BLOB(2K) xmlOutMessage;
 EXEC SQL END DECLARE SECTION;
   getMsgMaj = 2;
   getMsgMin = 0;

  EXEC SQL CALL SYSPROC.GET_MESSAGE(
                :getMsgMaj,
                :getMsgMin,
                'en_US',
                BLOB('
                <?xml version="1.0" encoding="UTF-8"?>
                <plist version="1.0">
                <dict>
                   <key>Document Type Name</key>
                   <string>
                   Data Server Message Input
                   </string>
                   <key>Document Type Major Version</key><integer>2</integer>
                   <key>Document Type Minor Version</key><integer>0</integer>
                   <key>Required Parameters</key>
                   <dict>
                      <key>SQLCODE</key><string>SQL1034</string>
                   </dict>
                </dict>
                </plist>'),
                null,
               :xmlOutput,
               :xmlOutMessage );
```

GET_SYSTEM_INFO procedure - Get system information

The GET_SYSTEM_INFO procedure returns information about the data server, including information about the system, the current instance, installed data server products, environment variables, available CPUs, and other system information.

Syntax



The schema is SYSPROC.

Procedure parameters

major_version

An input and output argument of type INTEGER that indicates the major document version. On input, this argument indicates the major document version that the caller supports for the XML documents passed as parameters in the procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error (+20458) if the version is not valid. On output, this parameter specifies the highest major document version that is supported by the procedure. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

If the XML document in the *xml_input* parameter specifies a Document Type Major Version key, the value for that key must be equal to the value provided in the *major_version* parameter, or an error (+20458) is raised.

Supported versions: 1

minor_version
An input and output argument of type INTEGER that indicates the minor document version. On input, this argument specifies the minor document version that the caller supports for the XML documents passed as parameters for this procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error if the version is not valid. On output, this parameter indicates the highest minor document version that is supported for the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

Supported versions: 0

requested_locale
An input argument of type VARCHAR(33) that specifies a locale. If the specified language is supported on the server, translated content is returned in the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names and values are not translated. The only translated portion of the XML output and XML message documents are the text for hint, display name, and display unit of each entry. The caller should always compare the requested language to the language that is used in the XML output document (see the document locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input
Currently, this procedure accepts no input. You must specify NULL for this parameter, or an error (+20458) is raised to indicate that the input is not valid.

xml_filter
An input argument of type BLOB(4K) that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see the topic that describes XPath filtering.

The following example selects the value for the Data Server Product Version from the XML output document: `/plist/dict/key[.='Data Server Product Version']/following-sibling::string`. If the key is not followed by the specified sibling, an error is returned.

xml_output
An output parameter of type BLOB(32MB) that returns a complete XML output document in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

The XML output document contains instance information, including information about the fix pack level, release, system information, and environment variables.

xml_message
An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about a SQL warning condition. This document is returned when a call to the procedure results in a SQL warning, and the warning message indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

Authorization

- SYSADM or DBADM authority
- EXECUTE privilege on the GET_SYSTEM_INFO procedure

Example

Example 1: Return the highest version of the procedure.

```
db2 "call sysproc.get_system_info(null,null,null,null,null,?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
  --------------------------
  Parameter Name  : MAJOR_VERSION
  Parameter Value : 1

  Parameter Name  : MINOR_VERSION
  Parameter Value : 0

  Parameter Name  : XML_OUTPUT
  Parameter Value : -

  Parameter Name  : XML_MESSAGE
  Parameter Value : -

  Return Status = 0
```

Example 2: Return system information.

```
 db2 "call sysproc.get_system_info(1,0,'en_US',null,null,?,?)"
```

The following is an example of output from this query:

```
  Value of output parameters
  --------------------------
  Parameter Name  : MAJOR_VERSION
  Parameter Value : 1

  Parameter Name  : MINOR_VERSION
  Parameter Value : 0

  Parameter Name  : XML_OUTPUT
  Parameter Value : x'3C3F786D6C20766572.....

  Parameter Name  : XML_MESSAGE
  Parameter Value : -

  Return Status = 0
```

The XML output document contains something similar to the following content:

```
<plist version="1.0">
<dict><key>Document Type Name</key><string>Data Server System Output</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
```

```
<key>Data Server Product Name</key><string>QDB2/AIX64</string>
<key>Data Server Product Version</key><string>9.7.0.0</string>
<key>Data Server Major Version</key><integer>9</integer>
<key>Data Server Minor Version</key><integer>7</integer>
<key>Data Server Platform</key><string>AIX 64BIT</string>
<key>Document Locale</key><string>en_US</string>
<key>Instance Information</key>
<dict>
   <key>Display Name</key><string>Instance Information</string>
   <key>Instance Name</key>
   <dict>
      <key>Display Name</key><string>Instance Name</string>
      <key>Value</key><string>myinstance</string>
      <key>Hint</key><string></string>
   </dict>
   <key>Partitionable State</key>
   <dict>
      <key>Display Name</key><string>Partitionable State</string>
      <key>Value</key><integer>0</integer>
      <key>Hint</key><string></string>
   </dict>
   <key>Number of Database Partitions</key>
   <dict>
      <key>Display Name</key><string>Number of Database Partitions</string>
      <key>Value</key><integer>1</integer>
      <key>Hint</key><string></string>
   </dict>
   .
   .
   .
</dict>
<key>Product Information</key>
<dict>
   <key>Display Name</key><string>Product Information</string>
   .
   .
   .
   <key>DB2_ENTERPRISE_SERVER_EDITION</key>
   <dict>
      <key>Display Name</key><string>DB2_ENTERPRISE_SERVER_EDITION</string>
      <key>Product short name</key>
      <dict>
         <key>Display Name</key><string>Product short name</string>
         <key>Value</key><string>ESE</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Licence</key>
      <dict>
         <key>Display Name</key><string>Licence</string>
         <key>Value</key><string>Y</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Product Release</key>
      <dict>
         <key>Display Name</key><string>Product Release</string>
         <key>Value</key><string>9.7</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Licence type</key>
      <dict>
         <key>Display Name</key><string>Licence type</string>
         <key>Value</key><string>DEVELOPER</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Hint</key><string></string>
   </dict>
   .
```

```
   .
   .
<key>Operating System Information</key>
<dict>
   <key>Display Name</key><string>Operating System Information</string>
   <key>Name</key>
   <dict>
      <key>Display Name</key><string>Name</string>
      <key>Value</key><string>AIX</string>
      <key>Hint</key><string></string>
   </dict>
   <key>Version</key>
   <dict>
      <key>Display Name</key><string>Version</string>
      <key>Value</key><string>5</string>
      <key>Hint</key><string></string>
   </dict>
   <key>Release</key>
   <dict>
      <key>Display Name</key><string>Release</string>
      <key>Value</key><string>3</string>
      <key>Hint</key><string></string>
   </dict>
   <key>Hostname</key>
   <dict>
      <key>Display Name</key><string>Hostname</string>
      <key>Value</key><string>achilles</string>
      <key>Hint</key><string></string>
   </dict>
   .
   .
   .
</dict>
<key>Workload Management Configuration</key>
<dict>
   <key>Display Name</key><string>Workload Management Configuration</string>
   <key>Service Class Information</key>
   <dict>
      <key>Display Name</key><string>Service Class Information</string>
      <key>1</key>
      <dict>
         <key>Display Name</key><string>1</string>
         <key>Service Class Name</key>
         <dict>
            <key>Display Name</key><string>Service Class Name</string>
            <key>Value</key><string>SYSDEFAULTSYSTEMCLASS</string>
            <key>Hint</key><string></string>
      </dict>
      <key>Parent Identifier</key>
      <dict>
         <key>Display Name</key><string>Parent Identifier</string>
         <key>Value</key><integer>0</integer>
         <key>Hint</key><string></string>
      </dict>
      <key>Parent Class Name</key>
      <dict>
         <key>Display Name</key><string>Parent Class Name</string>
         <key>Value</key><string></string>
         <key>Hint</key><string></string>
      </dict>
      <key>Creation Time</key>
      <dict>
         <key>Display Name</key><string>Creation Time</string>
         <key>Value</key><string>2008-04-21-15.14.32.956930</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Alter Time</key>
```

```
   <dict>
      <key>Display Name</key><string>Alter Time</string>
      <key>Value</key><string>2008-04-21-15.14.32.956930</string>
      <key>Hint</key><string></string>
   </dict>
   <key>Enabled</key>
   <dict>
      <key>Display Name</key><string>Enabled</string>
      <key>Value</key><string>Y</string>
      <key>Hint</key><string></string>
   </dict>
   <key>Agent Priority</key>
   <dict>
      <key>Display Name</key><string>Agent Priority</string>
      <key>Value</key><integer>-32768</integer>
      <key>Hint</key><string></string>
   </dict>
   <key>Prefetcher Priority</key>
   <dict>
      <key>Display Name</key><string>Prefetcher Priority</string>
      <key>Value</key><string> </string>
      <key>Hint</key><string></string>
   </dict>
   .
   .
   .
</dict>
.
.
.

<key>Workload Information</key>
<dict>
   <key>Display Name</key><string>Workload Information</string>
   <key>1</key>
   <dict>
      <key>Display Name</key><string>1</string>
      <key>Workload Name</key>
      <dict>
         <key>Display Name</key><string>Workload Name</string>
         <key>Value</key><string>SYSDEFAULTUSERWORKLOAD</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Evaluation Order</key>
      <dict>
         <key>Display Name</key><string>Evaluation Order</string>
         <key>Value</key><integer>1</integer>
         <key>Hint</key><string></string>
      </dict>
      <key>Creation Time</key>
      <dict>
         <key>Display Name</key><string>Creation Time</string>
         <key>Value</key><string>2008-04-21-15.14.32.955296</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Alter Time</key>
      <dict>
         <key>Display Name</key><string>Alter Time</string>
         <key>Value</key><string>2008-04-21-15.14.32.955296</string>
         <key>Hint</key><string></string>
      </dict>
      <key>Enabled</key>
      <dict>
         <key>Display Name</key><string>Enabled</string>
         <key>Value</key><string>Y</string>
         <key>Hint</key><string></string>
      </dict>
```

```
            <key>Allow Access</key>
            <dict>
               <key>Display Name</key><string>Allow Access</string>
               <key>Value</key><string>Y</string>
               <key>Hint</key><string></string>
            </dict>
            <key>Service Class Name</key>
            <dict>
               <key>Display Name</key><string>Service Class Name</string>
               <key>Value</key><string>SYSDEFAULTSUBCLASS</string>
               <key>Hint</key><string></string>
            </dict>
            <key>Parent Service Class Name</key>
            <dict>
               <key>Display Name</key><string>Parent Service Class Name</string>
               <key>Value</key><string>SYSDEFAULTUSERCLASS</string>
               <key>Hint</key><string></string>
            </dict>
            .
            .
            .
         </dict>
         <key>Hint</key><string></string>
      </dict>
   </dict>
</dict></dict></dict></plist>
```

Example 3: Call the GET_SYSTEM_INFO procure and pass in an unsupported locale.

```
db2 "call sysproc. get_system_info(1,0,'ja_JP',null,null,?,?)"
```

The following is an example of output from this query:

```
  Value of output parameters
  --------------------------
  Parameter Name  : MAJOR_VERSION
  Parameter Value : 1

  Parameter Name  : MINOR_VERSION
  Parameter Value : 0

  Parameter Name  : XML_OUTPUT
  Parameter Value : x'3C3F786D6C20766572.....

  Parameter Name  : XML_MESSAGE
  Parameter Value : -

  Return Status = 0

  SQL20461W  The procedure "SYSPROC.GET_SYSTEM_INFO" returned output in the
alternate locale, "en_US", instead of the locale, "ja_JP", specified in
parameter "3". SQLSTATE=01H57
```

The XML output document will contain the same content that is shown for Example 2.

Example 4: Call the procedure from a function.

```
 EXEC SQL BEGIN DECLARE SECTION;
  sqlint16  getSysInfMaj;
  sqlint16  getSysInfMin;

  SQL TYPE IS BLOB(2M) xmlOutput;
  SQL TYPE IS BLOB(2K) xmlOutMessage;
 EXEC SQL END DECLARE SECTION;
   getSysInfMaj = 1;
```

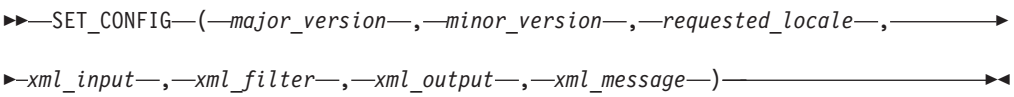
```
 getSysInfMin = 0;

EXEC SQL CALL SYSPROC.GET_SYSTEM_INFO(
              :getSysInfMaj,
              :getSysInfMin,
              'en_US',
               null,
               null,
              :xmlOutput,
              :xmlOutMessage );
```

SET_CONFIG procedure - Set configuration parameters

The SET_CONFIG stored procedure updates the database and database manager configuration parameters that are returned by the GET_CONFIG procedure.

The SET_CONFIG procedure accepts an input XML document that contains configuration parameters and their values, uses this information to update the specified configuration parameters, and returns an output XML document that indicates the update status of each configuration parameter.

Syntax



The schema is SYSPROC.

Procedure parameters

major_version
An input and output argument of type INTEGER that indicates the major document version. On input, this argument indicates the major document version that the caller supports for the XML documents passed as parameters in the procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error (+20458) if the version is not valid. On output, this parameter specifies the highest major document version that is supported by the procedure. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

If the XML document in the *xml_input* parameter specifies a Document Type Major Version key, the value for that key must be equal to the value provided in the *major_version* parameter, or an error (+20458) is raised.

Supported versions: 1

minor_version
An input and output argument of type INTEGER that indicates the minor document version. On input, this argument specifies the minor document version that the caller supports for the XML documents passed as parameters for this procedure (see the parameter descriptions for *xml_input*, *xml_output*, and *xml_message*). The procedure processes all XML documents in the specified version, or returns an error if the version is not valid. On output, this parameter indicates the highest minor document version that is supported for

the highest supported major version. To determine the highest supported document version, specify NULL for this input parameter and all other required parameters.

If the XML document in the *xml_input* parameter specifies a Document Type Minor Version key, the value for that key must be equal to the value provided in the *minor_version* parameter, or an error (+20458) is raised.

Supported versions: 0

requested_locale
An input argument of type VARCHAR(33) that specifies a locale. If the specified language is supported on the server, translated content is returned in the *xml_output* and *xml_message* parameters. Otherwise, content is returned in the default language. Only the language and possibly the territory information is used from the locale. The locale is not used to format numbers or influence the document encoding. For example, key names and values are not translated. The only translated portion of the XML output and XML message documents are the text for hint, display name, and display unit of each entry. The caller should always compare the requested language to the language that is used in the XML output document (see the document locale entry in the XML output document).

Currently, the only supported value for *requested_locale* is en_US.

xml_input
An input argument of type BLOB(32MB) that specifies an XML input document (encoded in UTF-8) that contains input values for the procedure.

For this procedure, the XML input document contains database and database manager configuration settings.

xml_filter
An input argument of type BLOB(4K) that specifies a valid XPath query string. Use a filter when you want to retrieve a single value from an XML output document. For more information, see the topic that describes XPath filtering.

The following example selects the value for a specific configuration parameter setting from the XML output document: `/plist/dict/key[.="Database Manager Configuration Parameter Settings"]/following-sibling::dict[1]/key[3]/following-sibling::dict[1]/dict[1]/key[.="Value"]/following-sibling::string[1]`. If the key is not followed by the specified sibling, an error is returned.

xml_output
An output parameter of type BLOB(32MB) that returns a complete XML output document in UTF-8. If a filter is specified, this parameter returns a string value. If the stored procedure is unable to return a complete output document (for example, if a processing error occurs that results in an SQL warning or error), this parameter is set to NULL.

When this procedure operates in *complete* mode, this parameter returns an XML document that contains the current configuration values set in the server. You can modify this document and pass it back to the procedure as the *xml_input* parameter. This approach provides a programmatic way to create valid XML input documents.

xml_message
An output parameter of type BLOB(64K) that returns a complete XML output document of type Data Server Message in UTF-8 that provides detailed information about a SQL warning condition. This document is returned when a

call to the procedure results in a SQL warning, and the warning message indicates that additional information is returned in the XML message output document. If the warning message does not indicate that additional information is returned, then this parameter is set to NULL.

Authorization

- SYSADM or DBADM authority
- EXECUTE privilege on the SET_CONFIG procedure

Example

Example 1: Return the latest version of the procedure.

```
db2 "call  sysproc.set_config (null,null,null,null,null,?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : -

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

Example 2: Run a script called `setconfig.sql` that updates a few database and database manager configuration parameters.

`setconfig.sql`:

```
call sysproc.set_config(1,0,'en_US',blob('
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Document Type Name</key><string>Data Server Set Configuration Input</string>
  <key>Document Type Major Version</key><integer>1</integer>
  <key>Document Type Minor Version</key><integer>0</integer>
  <key>Document Locale</key><string>en_US</string>
  <key>Database Manager Configuration Parameter Settings</key>
  <dict>
    <key>diaglevel</key><dict><key>Parameter Value</key>
    <dict>
      <key>Value</key><string>4</string>
    </dict>
  </dict>
  <key>fcm_num_buffers</key>
  <dict>
    <key>Parameter Value</key>
    <dict>
      <key>Value</key><string>4096</string>
    </dict>
    <key>Value Flags</key>
    <dict>
      <key>Value</key><string>MANUAL</string>
    </dict>
  </dict>
  <key>instance_memory</key>
```

```
  <dict>
    <key>Deferred Value</key>
    <dict>
      <key>Value</key><string>7424</string>
    </dict>
    <key>Deferred Value Flags</key>
    <dict>
        <key>Value</key><string>AUTOMATIC</string>
    </dict>
  </dict>
  </dict>
  <key>Database Partition</key>
  <dict>
    <key>All</key>
    <dict>
      <key>Database Configuration Parameter Settings</key>
      <dict>
        <key>avg_appls</key>
        <dict>
          <key>Parameter Value</key>
          <dict>
            <key>Value></key><string>2</string>
          </dict>
          <key>Value Flags</key>
          <dict>
            <key>Value</key><string>AUTOMATIC</string>
          </dict>
        </dict>
        <key>database_memory</key>
        <dict>
          <key>Deferred Value</key>
          <dict>
            <key>Value</key><string>2</string>
          </dict>
          <key>Deferred Value Flags</key>
          <dict>
            <key>Value</key><string>MANUAL</string>
          </dict>
        </dict>
      </dict>
    </dict>
  </dict>
</dict>
</plist>'), null, ?,?)@
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'3C3F78...'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

The output XML document contains something similar to the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
```

```
<key>Document Type Name</key>
<string>Data Server Set Configuration Output</string>
<key>Document Type Major Version</key><integer>1</integer>
<key>Document Type Minor Version</key><integer>0</integer>
<key>Data Server Product Name</key><string>QDB2/AIX64</string>
<key>Data Server Product Version</key><string>9.7.0.0</string>
<key>Data Server Major Version</key><integer>9</integer>
<key>Data Server Minor Version</key><integer>7</integer>
<key>Data Server Platform</key><string>AIX 64BIT</string>
<key>Document Locale</key><string>en_US</string>
<key>Database Manager Configuration Parameter Settings</key>
<dict>
  <key>Display Name</key>
  <string>Database Manager Configuration Parameter Settings</string>
  <key>diaglevel</key>
  <dict>
    <key>Display Name</key><string>diaglevel</string>
    <key>Parameter Value</key>
    <dict>
      <key>Display Name</key><string>Parameter Value</string>
      <key>Value</key><string>4</string>
    </dict>
    <key>Parameter Update Status</key>
    <dict>
      <key>Display Name</key><string>Parameter Update Status</string>
      <key>SQLCODE</key>
      <dict
        <key>Display Name</key><string>SQLCODE</string>
        <key>Value</key><integer>0</integer>
      </dict>
    <key>Message Tokens</key>
    <dict>
      <key>Display Name</key><string>Message Tokens</string>
      <key>Value</key><array><string></string></array>
    </dict>
    <key>SQLSTATE</key>
    <dict>
      <key>Display Name</key><string>SQLSTATE</string>
      <key>Value</key><string></string>
    </dict>
  </dict>
</dict>
<key>fcm_num_buffers</key>
<dict>
  <key>Display Name</key><string>fcm_num_buffers</string>
  <key>Parameter Value</key>
    <dict>
      <key>Display Name</key><string>Parameter Value</string>
      <key>Value</key><string>4096</string>
    </dict>
    <key>Value Flags</key>
    <dict>
      <key>Display Name</key><string>Value Flags</string>
      <key>Value</key><string>MANUAL</string>
    </dict>
    <key>Parameter Update Status</key>
    <dict>
      <key>Display Name</key><string>Parameter Update Status</string>
      <key>SQLCODE</key><dict>
      <key>Display Name</key><string>SQLCODE</string>
      <key>Value</key> <integer>0</integer>
    </dict>
    <key>Message Tokens</key>
    <dict>
      <key>Display Name</key><string>Message Tokens</string>
      <key>Value</key><array><string></string></array>
    </dict>
```

```
    <key>SQLSTATE</key>
    <dict>
      <key>Display Name</key><string>SQLSTATE</string>
      <key>Value</key><string></string>
    </dict>
  </dict>
</dict>
<key>instance_memory</key>
<dict>
  <key>Display Name</key><string>instance_memory</string>
  <key>Deferred Value</key>
  <dict>
    <key>Display Name</key><string>Deferred Value</string>
   <key>Value</key><string>7424</string>
  </dict>
  <key>Deferred Value Flags</key>
  <dict>
    <key>Display Name</key><string>Deferred Value Flags</string>
    <key>Value</key><string>AUTOMATIC</string>
  </dict>
  <key>Parameter Update Status</key>
  <dict>
    <key>Display Name</key><string>Parameter Update Status</string>
    <key>SQLCODE</key>
    <dict>
      <key>Display Name</key><string>SQLCODE</string>
      <key>Value</key><integer>0</integer>
    </dict>
      <key>Message Tokens</key>
      <dict>
        <key>Display Name</key><string>Message Tokens</string>
        <key>Value</key><array><string></string></array>
      </dict>
      <key>SQLSTATE</key>
       <dict>
         <key>Display Name</key><string>SQLSTATE</string>
         <key>Value</key><string></string>
       </dict>
     </dict>
   </dict>
  </dict>
  <key>Database Partition</key>
  <dict>
    <key>Display Name</key><string>Database Partition</string>
    <key>All</key>
    <dict>
      <key>Display Name</key><string>All</string>
     <key>Database Configuration Parameter Settings</key>
     <dict>
       <key>Display Name</key>
       <string>Database Configuration Parameter Settings</string>
       <key>avg_appls</key>
       <dict>
         <key>Display Name</key><string>avg_appls</string>
         <key>Parameter Value</key>
         <dict>
           <key>Display Name</key><string>Parameter Value</string>
           <key>Value</key><string>2</string>
         </dict>
        <key>Value Flags</key>
        <dict>
          <key>Display Name</key><string>Value Flags</string>
          <key>Value</key><string>AUTOMATIC</string>
        </dict>
        <key>Parameter Update Status</key>
        <dict>
          <key>Display Name</key><string>Parameter Update Status</string>
```

```
            <key>Update Coverage</key>
            <dict>
              <key>Display Name</key><string>Update Coverage</string>
              <key>Value</key><string>Complete</string>
            </dict>
            <key>SQLCODE</key>
            <dict>
              <key>Display Name</key><string>SQLCODE</string>
              <key>Value</key><integer>0</integer>
            </dict>
            <key>Message Tokens</key>
            <dict>
              <key>Display Name</key><string>Message Tokens</string>
              <key>Value</key><array><string></string> </array>
            </dict>
            <key>SQLSTATE</key>
            <dict>
               <key>Display Name</key><string>SQLSTATE</string>
               <key>Value</key><string></string>
            </dict>
          </dict>
        </dict>
        <key>database_memory</key>
        <dict>
          <key>Display Name</key><string>database_memory</string>
          <key>Deferred Value</key>
          <dict>
            <key>Display Name</key><string>Deferred Value</string>
            <key>Value</key><string>2</string>
          </dict>
          <key>Deferred Value Flags</key>
          <dict>
            <key>Display Name</key><string>Deferred Value Flags</string>
            <key>Value</key><string>MANUAL</string>
          </dict>
          <key>Parameter Update Status</key>
          <dict>
            <key>Display Name</key><string>Parameter Update Status</string>
            <key>Update Coverage</key>
            <dict>
              <key>Display Name</key><string>Update Coverage</string>
              <key>Value</key><string>Complete</string>
            </dict>
            <key>SQLCODE</key>
            <dict>
              <key>Display Name</key><string>SQLCODE</string>
              <key>Value</key><integer>0</integer>
            </dict>
            <key>Message Tokens</key>
            <dict>
              <key>Display Name</key><string>Message Tokens</string>
              <key>Value</key><array><string></string></array>
            </dict>
            <key>SQLSTATE</key>
            <dict>
              <key>Display Name</key><string>SQLSTATE</string>
              <key>Value</key><string></string>
            </dict>
          </dict>
        </dict>
      </dict>
    </dict>
  </dict>
</dict>
</plist>
```

Example 3: Specify a filter to return the value for a specific configuration parameter.

```
db2 "call sysproc.set_config(1,0, 'en_US', blob('
<plist version="1.0">
<dict>
   <key>Document Type Name</key><string>Data Server Set Configuration Input</string>
   <key>Document Type Major Version</key><integer>1</integer>
   <key>Document Type Minor Version</key><integer>0</integer>
   <key>Document Locale</key><string>en_US</string>
   <key>Database Manager Configuration Parameter Settings</key>
   <dict>
      <key>diaglevel</key>
      <dict>
         <key>Parameter Value</key>
         <dict>
            <key>Value</key><string>4</string>
         </dict>
      </dict>
      <key>fcm_num_buffers</key>
      <dict>
         <key>Parameter Value</key>
         <dict>
            <key>Value</key><string>4096</string>
         </dict>
        <key>Value Flags</key>
        <dict>
           <key>Value</key><string>MANUAL</string>
        </dict>
      </dict>
      <key>instance_memory</key>
      <dict>
         <key>Deferred Value</key>
         <dict>
            <key>Value</key><string>7424</string>
         </dict>
         <key>Deferred Value Flags</key>
         <dict>
            <key>Value</key><string>AUTOMATIC</string>
         </dict>
      </dict>
   </dict>
   <key>Database Partition</key>
   <dict>
      <key>All</key>
      <dict>
         <key>Database Configuration Parameter Settings</key>
         <dict>
            <key>avg_appls</key>
            <dict>
               <key>Parameter Value</key>
               <dict>
                  <key>Value></key><string>2</string>
               </dict>
               <key>Value Flags</key>
               <dict>
                  <key>Value</key><string>AUTOMATIC</string>
               </dict>
            </dict>
            <key>database_memory</key>
            <dict>
               <key>Deferred Value</key>
               <dict>
                  <key>Value</key><string>2</string>
               </dict>
               <key>Deferred Value Flags</key>
               <dict>
                  <key>Value</key><string>MANUAL</string>
               </dict>
            </dict>
```

```
        </dict>
      </dict>
   </dict>
</dict>
</plist>'),
blob('/plist/dict/key[.="Database Manager Configuration Parameter Settings"]
/following-sibling::dict[1]/key[3]
/following-sibling::dict[1]/dict[1]/key[.="Value"]
/following-sibling::string[1]'),?,?)"
```

The following is an example of output from this query:

```
Value of output parameters
--------------------------
Parameter Name : MAJOR_VERSION
Parameter Value : 1

Parameter Name : MINOR_VERSION
Parameter Value : 0

Parameter Name : XML_OUTPUT
Parameter Value : x'34303936'

Parameter Name : XML_MESSAGE
Parameter Value : -

Return Status = 0
```

The following value is returned for *xml_output*:

```
4096
```

Example 4: Call the procedure from a function.

```
  EXEC SQL BEGIN DECLARE SECTION;
  sqlint16  getconfigMaj;
  sqlint16  getconfigMin;

  SQL TYPE IS BLOB(2M) xmlOutput;
  SQL TYPE IS BLOB(2K) xmlOutMessage;
  EXEC SQL END DECLARE SECTION;
   getconfigMaj = 1;
   getconfigMin = 0;

  EXEC SQL CALL SYSPROC.SET_CONFIG(
                :getconfigMaj,
                :getconfigMin,
                'en_US',
                BLOB('blob('
                <?xml version="1.0" encoding="UTF-8"?>
                <plist version="1.0">
                <dict>
                  <key>Document Type Name</key>
                  <string>Data Server Set Configuration Input</string>
                  <key>Document Type Major Version</key><integer>1</integer>
                  <key>Document Type Minor Version</key><integer>0</integer>
                  <key>Document Locale</key><string>en_US</string>
                  <key>Database Manager Configuration Parameter Settings</key>
                  <dict>
                    <key>diaglevel</key><dict><key>Parameter Value</key>
                    <dict>
                      <key>Value</key><string>4</string>
                    </dict>
                  </dict>
                  <key>fcm_num_buffers</key>
                  <dict>
                    <key>Parameter Value</key>
                    <dict>
```

```
      <key>Value</key><string>4096</string>
    </dict>
    <key>Value Flags</key>
    <dict>
      <key>Value</key><string>MANUAL</string>
    </dict>
  </dict>
  <key>instance_memory</key>
  <dict>
    <key>Deferred Value</key>
    <dict>
      <key>Value</key><string>7424</string>
    </dict>
    <key>Deferred Value Flags</key>
    <dict>
        <key>Value</key><string>AUTOMATIC</string>
    </dict>
  </dict>
  </dict>
  <key>Database Partition</key>
  <dict>
    <key>All</key>
    <dict>
      <key>Database Configuration Parameter Settings</key>
      <dict>
        <key>avg_appls</key>
        <dict>
          <key>Parameter Value</key>
          <dict>
            <key>Value></key><string>2</string>
          </dict>
          <key>Value Flags</key>
          <dict>
            <key>Value</key><string>AUTOMATIC</string>
          </dict>
        </dict>
        <key>database_memory</key>
        <dict>
          <key>Deferred Value</key>
          <dict>
            <key>Value</key><string>2</string>
          </dict>
          <key>Deferred Value Flags</key>
          <dict>
            <key>Value</key><string>MANUAL</string>
          </dict>
        </dict>
      </dict>
    </dict>
  </dict>
</dict>
</plist>'),
null,
:xmlOutput,
:xmlOutMessage );
```

Chapter 9. Configuration routines and views

DB_PARTITIONS

The DB_PARTITIONS table function returns the contents of the db2nodes.cfg file in table format.

Syntax

The schema is SYSPROC.

Authorization

EXECUTE privilege on the DB_PARTITIONS table function.

Table function parameters

The function has no input parameters.

Example

Retrieve information from a 3 logical partition database.

```
SELECT * FROM TABLE(DB_PARTITIONS()) AS T
```

The following is an example of output from this query.

```
PARTITION_NUMBER HOST_NAME                ...- PORT_NUMBER SWITCH_NAME
---------------- ---------------------    ...- ----------- -----------
               0 jessicae.torolab.ibm.com                 0 jessicae
               1 jessicae.torolab.ibm.com                 1 jessicae
               2 jessicae.torolab.ibm.com                 2 jessicae

  3 record(s) selected.
```

Information returned

Table 84. Information returned by the DB_PARTITIONS table function

Column name	Data type	Description
PARTITION_NUMBER	SMALLINT	A unique number between 0 and 999 that identifies a database partition server in a partitioned database environment.
HOST_NAME	VARCHAR(128)	The TCP/IP host name of the database partition server.
PORT_NUMBER	SMALLINT	The port number for the database partition server.

Table 84. Information returned by the DB_PARTITIONS table function (continued)

Column name	Data type	Description
SWITCH_NAME	VARCHAR(128)	The name of a high speed interconnect, or switch, for database partition communications.

DBCFG administrative view - Retrieve database configuration parameter information

The DBCFG administrative view retrieves database configuration parameter information for the currently connected database for all database partitions.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the DBCFG administrative view
- CONTROL privilege on the DBCFG administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the DB_GET_CFG table function
- DATAACCESS authority

Examples

Example 1: Retrieve the automatic maintenance settings in the database configuration that are stored in memory for all database partitions.

```
SELECT DBPARTITIONNUM, NAME, VALUE FROM SYSIBMADM.DBCFG WHERE NAME LIKE 'auto_%'
```

The following is an example of output for this query.

```
DBPARTITIONNUM NAME                            VALUE
-------------- ------------------------------- --------------
             0 auto_maint                      OFF
             0 auto_db_backup                  OFF
             0 auto_tbl_maint                  OFF
             0 auto_runstats                   OFF
             0 auto_stats_prof                 OFF
             0 auto_prof_upd                   OFF
             0 auto_reorg                      OFF
             0 autorestart                     ON

  8 record(s) selected.
```

Example 2: Retrieve all the database configuration parameters values stored on disk for all database partitions.

```
SELECT NAME, DEFERRED_VALUE, DBPARTITIONNUM FROM SYSIBMADM.DBCFG
```

The following is an example of output for this query.

```
NAME                DEFERRED_VALUE      DBPARTITIONNUM
----------------...- ---------------...- --------------
app_ctl_heap_sz     128                              0
```

```
appgroup_mem_sz      30000                          0
applheapsz           256                            0
archretrydelay       20                             0
...
autorestart          ON                             0
avg_appls            1                              0
blk_log_dsk_ful      NO                             0
catalogcache_sz      -1                             0
...
```

Information returned

Table 85. Information returned by the DBCFG administrative view

Column name	Data type	Description
NAME	VARCHAR(32)	Configuration parameter name.
VALUE	VARCHAR(1024)	The current value of the configuration parameter stored in memory.
VALUE_FLAGS	VARCHAR(10)	Provides specific information for the configuration parameter current value. Valid values are: • NONE - no additional information • AUTOMATIC - the configuration parameter has been set to automatic
DEFERRED_VALUE	VARCHAR(1024)	The value of the configuration parameter on disk. For some database configuration parameters, changes only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) The changes take effect at the next connection to the database.
DEFERRED_VALUE_FLAGS	VARCHAR(10)	Provides specific information for the configuration parameter deferred value. Valid values are: • NONE - no additional information • AUTOMATIC - the configuration parameter has been set to automatic
DATATYPE	VARCHAR(128)	Configuration parameter data type.
DBPARTITIONNUM	SMALLINT	Database partition number.

DBMCFG administrative view - Retrieve database manager configuration parameter information

The DBMCFG administrative view returns database manager configuration parameter information including the values in memory and the values stored on disk.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the DBMCFG administrative view
- CONTROL privilege on the DBMCFG administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the DBM_GET_CFG table function
- DATAACCESS authority

Examples

Example 1: Retrieve values for all the database manager configuration parameters stored on disk:

```
SELECT NAME, DEFERRED_VALUE FROM SYSIBMADM.DBMCFG
```

The following is an example of output for this query.

```
NAME                           DEFERRED_VALUE
------------------------------ ------------------...----
agent_stack_sz                 0
agentpri                       -1
alternate_auth_enc             AES_ONLY
aslheapsz                      15
audit_buf_sz                   0
authentication                 SERVER
catalog_noauth                 YES
clnt_krb_plugin
...
comm_bandwidth                 0.000000e+00
conn_elapse                    0
cpuspeed                       4.000000e-05
dft_account_str
dft_mon_bufpool                OFF
...
dft_mon_timestamp              ON
dft_mon_uow                    OFF
...
jdk_path                       /wsdb/v91/bldsupp/AIX5L
...
ssl_svcename                   22711
ssl_svr_keydb                  /GSKit/Keystore/key.kdb
ssl_svr_label
ssl_svr_stash                  /GSKit/Keystore/key.sth
```

Example 2: Retrieve all the database manager configuration parameters values.

```
SELECT * FROM SYSIBMADM.DBMCFG
```

The following is an example of output for this query.

```
NAME                VALUE           VALUE_FLAGS ...
------------...------------...- ------------ ...
agent_stack_sz      0               NONE        ...
agentpri            -1              NONE        ...
alternate_auth_enc  NOT_SPECIFIED   NONE
aslheapsz           15              NONE        ...
audit_buf_sz        0               NONE        ...
authentication      SERVER          NONE        ...
catalog_noauth      YES             NONE        ...
clnt_krb_plugin                     NONE        ...
clnt_pw_plugin                      NONE        ...
comm_bandwidth      0.000000e+00    NONE        ...
conn_elapse         0               NONE        ...
cpuspeed            4.000000e-05    NONE        ...
dft_account_str                     NONE        ...
dft_mon_bufpool     OFF             NONE        ...
dft_mon_lock        OFF             NONE        ...
dft_mon_sort        OFF             NONE        ...
dft_mon_stmt        OFF             NONE        ...
dft_mon_table       OFF             NONE        ...
...                                           ...
dir_cache           YES             NONE        ...
discover            SEARCH          NONE        ...
discover_inst       ENABLE          NONE        ...
fcm_num_anchors     0               AUTOMATIC   ...
fcm_num_buffers     0               AUTOMATIC   ...
fcm_num_connect     0               AUTOMATIC   ...
```

Output for this query (continued).

```
... DEFERRED_VALUE     DEFERRED_VALUE_FLAGS      DATATYPE
... -------------...- ------------------------ ---------...-
... 0                  NONE                      INTEGER
... -1                 NONE                      INTEGER
... AES_ONLY           NONE                      VARCHAR(32)
... 15                 NONE                      BIGINT
... 0                  NONE                      BIGINT
... SERVER             NONE                      VARCHAR(32)
... YES                NONE                      VARCHAR(3)
...                    NONE                      VARCHAR(32)
...                    NONE                      VARCHAR(32)
... 0.000000e+00       NONE                      REAL
... 0                  NONE                      INTEGER
... 4.000000e-05       NONE                      REAL
...                    NONE                      VARCHAR(25)
... OFF                NONE                      VARCHAR(3)
... OFF                NONE                      VARCHAR(3)
... OFF                NONE                      VARCHAR(3)
... OFF                NONE                      VARCHAR(3)
... OFF                NONE                      VARCHAR(3)
...
... YES                NONE                      VARCHAR(3)
... SEARCH             NONE                      VARCHAR(8)
... ENABLE             NONE                      VARCHAR(8)
... 0                  AUTOMATIC                 BIGINT
... 512                AUTOMATIC                 BIGINT
... 0                  AUTOMATIC                 BIGINT
...
```

Information returned

Table 86. Information returned by the DBMCFG administrative view

Column name	Data type	Description
NAME	VARCHAR(32)	Configuration parameter name.
VALUE	VARCHAR(256)	The current value of the configuration parameter stored in memory.
VALUE_FLAGS	VARCHAR(10)	Provides specific information for the configuration parameter current value. Valid values are: • NONE - no additional information • AUTOMATIC - the configuration parameter has been set to automatic
DEFERRED_VALUE	VARCHAR(256)	The value of the configuration parameter on disk. For some database manager configuration parameters, the database manager must be stopped (db2stop) and restarted (db2start) for this value to take effect.
DEFERRED_VALUE_FLAGS	VARCHAR(10)	Provides specific information for the configuration parameter deferred value. Valid values are: • NONE - no additional information • AUTOMATIC - the configuration parameter has been set to automatic
DATATYPE	VARCHAR(128)	Configuration parameter data type.

REG_VARIABLES administrative view - Retrieve DB2 registry settings in use

The REG_VARIABLES administrative view returns the DB2 registry settings from all database partitions. The DB2 registry variable values returned when the REG_VARIABLES administrative view is queried can differ from those returned by the db2set command if a DB2 registry variable is configured using the db2set command after the instance has been started. The difference occurs because REG_VARIABLES only returns the values that were in effect when the instance was started.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the REG_VARIABLES administrative view
- CONTROL privilege on the REG_VARIABLES administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the REG_LIST_VARIABLES table function
- DATAACCESS authority

Example

Request the DB2 registry settings that are currently being used.

```
SELECT * from SYSIBMADM.REG_VARIABLES
```

The following is an example of output from this query.

```
DBPARTITIONNUM REG_VAR_NAME        REG_VAR_VALUE     IS_AGGREGATE AGGREGATE_NAME
-------------- ---------------...- ------------...- ------------ --------------...-
             0 DB2ADMINSERVER      DB2DAS00                     0 -
             0 DB2INSTPROF         D:\SQLLIB                    0 -
             0 DB2PATH             D:\SQLLIB                    0 -
             0 DB2SYSTEM           D570                         0 -
             0 DB2TEMPDIR          D:\SQLLIB\                   0 -
             0 DB2_EXTSECURITY     YES                          0 -

  6 record(s) selected.
```

Information returned

Table 87. Information returned by the REG_VARIABLES administrative view

Column name	Data type	Description
DBPARTITIONNUM	SMALLINT	Logical partition number of each database partition on which the function operates.
REG_VAR_NAME	VARCHAR(256)	Name of the DB2 registry variable.
REG_VAR_VALUE	VARCHAR(2048)	Current setting of the DB2 registry variable.
IS_AGGREGATE	SMALLINT	Indicates whether or not the DB2 registry variable is an aggregate variable. The possible return values are 0 if it is not an aggregate variable, and 1 if it is an aggregate variable.

Table 87. Information returned by the REG_VARIABLES administrative view (continued)

Column name	Data type	Description
AGGREGATE_NAME	VARCHAR(256)	Name of the aggregate if the DB2 registry variable is currently obtaining its value from a configured aggregate. If the registry variable is not being set through an aggregate, or is set through an aggregate but has been overridden, the value of AGGREGATE_NAME is NULL.
LEVEL	CHAR(1)	Indicates the level at which the DB2 registry variable acquires its value. The possible return values and the corresponding levels that they represent are: • I = instance • G = global • N = database partition • E = environment

Chapter 10. Environment views

ENV_FEATURE_INFO administrative view - Return license information for DB2 features

The ENV_FEATURE_INFO administrative view returns information about all available features for which a license is required. For each feature, there is information about whether or not a valid license for the feature has been installed.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the ENV_FEATURE_INFO administrative view
- CONTROL privilege on the ENV_FEATURE_INFO administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the ENV_GET_FEATURE_INFO table function
- DATAACCESS authority

Example

Request the installed DB2 features license information.

```
SELECT * FROM SYSIBMADM.ENV_FEATURE_INFO
```

The following is an example of output from this query.

```
FEATURE_NAME  FEATURE_FULLNAME                                ...
------------  --------------------------------------------  ...
DPF           DB2_DATABASE_PARTITIONING_FEATURE             ...
POESE         DB2_PERFORMANCE_OPTIMIZATION_FEATURE_FOR_ESE  ...
SO            DB2_STORAGE_OPTIMIZATION_FEATURE              ...
AAC           DB2_ADVANCED_ACCESS_CONTROL_FEATURE           ...
GEO           DB2_GEODETIC_DATA_MANAGEMENT_FEATURE          ...
HFESE         IBM_HOMOGENEOUS_FEDERATION_FEATURE_FOR_ESE    ...
XMLESE        DB2_PUREXML_FEATURE_FOR_ESE                   ...
```

Output from this query (continued).

```
... LICENSE_INSTALLED  PRODUCT_NAME  FEATURE_USE_STATUS
... ----------------   ------------  ------------------
... Y                  ESE           IN_COMPLIANCE
... Y                  ESE           IN_COMPLIANCE
... Y                  ESE           IN_COMPLIANCE
... Y                  ESE           NOT_USED
... Y                  ESE           NOT_USED
... Y                  ESE           NOT_USED
... N                  ESE           IN_VIOLATION
```

ENV_FEATURE_INFO administrative view metadata

Table 88. ENV_FEATURE_INFO administrative view metadata

Column name	Data type	Description
FEATURE_NAME	VARCHAR(26)	Short names for DB2 features which are available on licensed DB2 servers.
FEATURE_FULLNAME	VARCHAR(100)	Full name of DB2 features. Column values will be displayed in English in uppercase. Words are separated with an underscore character instead of a space character.
LICENSE_INSTALLED	CHAR(1)	Indicates if feature is licensed. If the value is 'N', the feature is not licensed. If the value is 'Y', the feature is licensed.
PRODUCT_NAME	VARCHAR(26)	Identifiers for DB2 server product on which the feature is available. The possible return values are: • ESE - DB2 Enterprise Server Edition • WSE - DB2 Workgroup Server Edition • EXP - DB2 Express® Edition
FEATURE_USE_STATUS	VARCHAR(30)	Indicates the license compliance status. This value indicates the usage status of the feature. There are three possible values: • IN_COMPLIANCE: Feature has been used at least once and there is a valid license for the feature. • IN_VIOLATION: Feature has been used at least once and there is no valid license for the feature. • NOT_USED: Feature has not been used.

ENV_INST_INFO administrative view - Retrieve information about the current instance

The ENV_INST_INFO administrative view returns information about the current instance.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the ENV_INST_INFO administrative view
- CONTROL privilege on the ENV_INST_INFO administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the ENV_GET_INST_INFO table function
- DATAACCESS authority

Example

Request information about the current instance.

```
SELECT * FROM SYSIBMADM.ENV_INST_INFO
```

The following is an example of output for this query.

```
INST_NAME          IS_INST_PARTITIONABLE NUM_DBPARTITIONS INST_PTR_SIZE ...
-----------...---- --------------------- ---------------- ------------- ...
DB2                                    0                1            32 ...

  1 record(s) selected.
```

Output for this query (continued).

```
... RELEASE_NUM       SERVICE_LEVEL        BLD_LEVEL        PTF        FIXPACK_NUM
... -----------...--- ---------------...--- ----------...--- ----...--- -----------
... 01010107          DB2 v9.1.0.115       n051106                             0
```

Information returned

Table 89. Information returned by the ENV_INST_INFO administrative view

Column name	Data type	Description
INST_NAME	VARCHAR(128)	Name of the current instance.
IS_INST_PARTITIONABLE	SMALLINT	Indicates whether or not the current instance is a partitionable database server instance. Possible return values are 0 if it is not a partitionable database server instance, and 1 if it is a partitionable database server instance.
NUM_DBPARTITIONS	INTEGER	Number of database partitions. If it is not a partitioned database environment, returns a value of 1.
INST_PTR_SIZE	INTEGER	Bit size of the current instance (32 or 64).
RELEASE_NUM	VARCHAR(128)	Internal release number, as returned by the db2level command; for example, 03030106.
SERVICE_LEVEL	VARCHAR(128)	Service level, as returned by the db2level command; for example, DB2 v8.1.1.80.
BLD_LEVEL	VARCHAR(128)	Build level, as returned by the db2level command; for example, n041021.
PTF	VARCHAR(128)	Program temporary fix (PTF) identifier, as returned by the db2level command; for example, U498350.
FIXPACK_NUM	INTEGER	Fix Pack number, as returned by the db2level command; for example, 9.

ENV_PROD_INFO administrative view - Retrieve information about installed DB2 products

The ENV_PROD_INFO administrative view returns information about installed DB2 products.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the ENV_PROD_INFO administrative view
- CONTROL privilege on the ENV_PROD_INFO administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the ENV_GET_PROD_INFO_V95 table function
- DATAACCESS authority

Example

Request the installed DB2 product information.

```
SELECT * FROM SYSIBMADM.ENV_PROD_INFO
```

The following is an example of output from this query.

```
INSTALLED_PROD  INSTALLED_PROD_FULLNAME        ...
--------------  ------------------------------ ...
ESE             DB2_ENTERPRISE_SERVER_EDITION  ...
WSE             DB2_WORKGROUP_SERVER_EDITION   ...
EXP             DB2_EXPRESS_EDITION            ...
```

Output from this query (continued).

```
... LICENSE_INSTALLED  PROD_RELEASE  LICENSE_TYPE
... ----------------  ------------  ---------------------
... Y                 9.5           AUTHORIZED_USER_OPTION
... N                 9.5           LICENSE_NOT_REGISTERED
... Y                 9.5           RESTRICTED
```

ENV_PROD_INFO administrative view metadata

Table 90. ENV_PROD_INFO administrative view metadata

Column name	Data type	Description
INSTALLED_PROD	VARCHAR(26)	Identifiers for DB2 products that are installed on the system.
INSTALLED_PROD_FULLNAME	VARCHAR(100)	Full name of installed DB2 products. Column values will be displayed in English in uppercase. Words are separated with an underscore character.
LICENSE_INSTALLED	CHAR(1)	Indicates if product is licensed. If the value is N, the product is not licensed. If the value is Y, the product is licensed.
PROD_RELEASE	VARCHAR(26)	Product release number.

Table 90. ENV_PROD_INFO administrative view metadata (continued)

Column name	Data type	Description
LICENSE_TYPE	VARCHAR(50)	Name of the type of license that is installed for the product. The possible return values are: • 12_MONTHS_LICENSE_AND_SUBSCRIPTION • AUTHORIZED_USER • AUTHORIZED_USER_OPTION • CLIENT_DEVICE • CPU • CPU_OPTION • HOST_SERVER_AND_MSU • LICENSE_NOT_REGISTERED • MANAGED_PROCESSOR • N/A • RESTRICTED • TRIAL • UNWARRANTED • USER

ENV_SYS_INFO administrative view - Retrieve information about the system

The ENV_SYS_INFO administrative view returns information about the system.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the ENV_SYS_INFO administrative view
- CONTROL privilege on the ENV_SYS_INFO administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the ENV_GET_SYS_INFO table function
- DATAACCESS authority

Example

Request information about the system.

```
SELECT * from SYSIBMADM.ENV_SYS_INFO
```

The following is an example of output from this query.

```
OS_NAME      OS_VERSION     OS_RELEASE         HOST_NAME
--------...- ----------...- --------------...- ---------...-
WIN32_NT     5.1            Service Pack 1     D570

  1 record(s) selected.
```

Output from this query (continued).

```
... TOTAL_CPUS  CONFIGURED_CPUS TOTAL_MEMORY
... ----------- --------------- ------------
...           1               2         1527
```

Information returned

Table 91. Information returned by the ENV_SYS_INFO administrative view

Column name	Data type	Description
OS_NAME	VARCHAR(256)	Name of the operating system.
OS_VERSION	VARCHAR(256)	Version number of the operating system.
OS_RELEASE	VARCHAR(256)	Release number of the operating system.
HOST_NAME	VARCHAR(256)	Name of the system.
TOTAL_CPUS	INTEGER	Total number of physical CPUs on the system.
CONFIGURED_CPUS	INTEGER	Number of configured physical CPUs on the system.
TOTAL_MEMORY	INTEGER	Total amount of memory on the system (in megabytes).

ENV_SYS_RESOURCES administrative view - Return system information

The ENV_SYS_RESOURCES administrative view returns operating system, CPU, memory and other information related to the system.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the ENV_SYS_RESOURCES administrative view
- CONTROL privilege on the ENV_SYS_RESOURCES administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the ENV_GET_SYS_RESOURCES table function
- DATAACCESS authority

Example

```
SELECT SUBSTR(NAME,1,20) AS NAME, SUBSTR(VALUE,1,10) AS VALUE,
  SUBSTR(DATATYPE,1,10) AS DATATYPE, DBPARTITIONNUM
FROM SYSIBMADM.ENV_SYS_RESOURCES
WHERE SUBSTR(NAME,1,8)='CPU_LOAD' OR NAME='CPU_USAGE_TOTAL'
```

The following is an example of output from this query.

```
NAME                 VALUE      DATATYPE   DBPARTITIONNUM
-------------------- ---------- ---------- --------------
CPU_LOAD_SHORT       0.044052   DECIMAL                 0
```

```
CPU_LOAD_MEDIUM         0.087250   DECIMAL                0
CPU_LOAD_LONG           0.142059   DECIMAL                0
CPU_USAGE_TOTAL         7          SMALLINT               0

  4 record(s) selected.
```

ENV_SYS_RESOURCES administrative view metadata

Table 92. ENV_SYS_RESOURCES administrative view metadata

Column name	Data type	Description
NAME	VARCHAR(128)	Name of the attribute. See Table 93 for possible values. **Note:** Some attributes might not be available depending on the operating system and hardware configuration at the server.
VALUE	VARCHAR(1024)	The value of the attribute.
DATATYPE	VARCHAR(128)	Attribute data type.
UNIT	VARCHAR(128)	Unit used for the VALUE column if applicable. NULL is returned if not applicable.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

Table 93. Possible values for the NAME column

Information type	Name	Data Types	Description	Platforms that return this information	UNIT
Operating system	OS_NAME	VARCHAR(256)	Name of the operating system software.	All	NULL
	HOST_NAME	VARCHAR(256)	Host name of the system.	All	NULL
	OS_VERSION	VARCHAR(256)	Version of the operating system. For example, AIX: 4.3 version = 4.	All	NULL
	OS_RELEASE	VARCHAR(256)	Release of the operating system. For example, AIX: 4.3 release = 3.	All	NULL
	MACHINE_IDENTIFICATION	VARCHAR(256)	Machine hardware identification.	All	NULL
	OS_LEVEL	VARCHAR(256)	Maintenance level of the current version and release. For example, LINUX: 2.4.9, level = 9.	Linux	NULL

Table 93. Possible values for the NAME column (continued)

Information type	Name	Data Types	Description	Platforms that return this information	UNIT
CPU	CPU_TOTAL	BIGINT	Total number of CPUs.	All	NULL
	CPU_ONLINE	BIGINT	Number of CPUs online.	All	NULL
	CPU_CONFIGURED	BIGINT	Number of CPUs configured.	All	NULL
	CPU_SPEED	BIGINT	Speed of CPUs.	All	MHz
	CPU_TIMEBASE	BIGINT	Frequency of timebase register increment.	Linux PowerPC®	Hz
	CPU_HMT_DEGREE	BIGINT	On systems that support hardware multithreading (HMT), this is the number of processors that a physical processor will appear to the operating system as. On non-HMT systems, this value is 1. On HMT systems, "total" will reflect the number of logical CPUs. To get the number of physical CPUs, divide the "total" by "threadingDegree".	All	NULL
	CPU_CORES_PER_SOCKET	BIGINT	Number of CPU cores per socket. On single core systems this value is 1.	All	NULL
Physical memory	MEMORY_TOTAL	BIGINT	Total size of physical memory.	All	MB
	MEMORY_FREE	BIGINT	Amount of free physical memory.	All	MB
	MEMORY_SWAP_TOTAL	BIGINT	Total amount of swap space.	All	MB
	MEMORY_SWAP_FREE	BIGINT	Amount of free swap space.	All	MB
Virtual memory	VIRTUAL_MEM_TOTAL	BIGINT	Total amount of virtual memory on the system.	All	MB
	VIRTUAL_MEM_RESERVED	BIGINT	Amount of reserved virtual memory.	All	MB
	VIRTUAL_MEM_FREE	BIGINT	Amount of virtual memory free.	All	MB

Table 93. Possible values for the NAME column (continued)

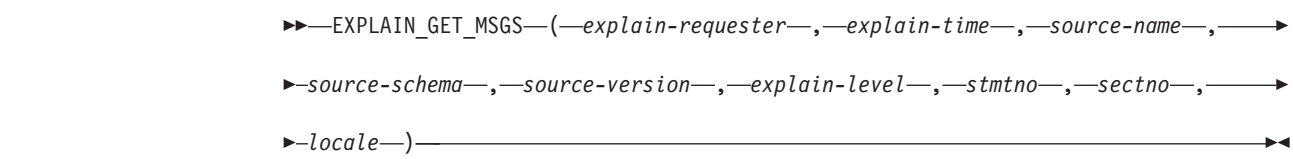
Information type	Name	Data Types	Description	Platforms that return this information	UNIT
CPU load	CPU_LOAD_SHORT	DECIMAL	Shortest period duration. For example, load samples over last 5 minutes.	All except Windows operating systems	NULL
	CPU_LOAD_MEDIUM	DECIMAL	Medium period duration. For example, load samples over last 10 minutes.	All except Windows operating systems	NULL
	CPU_LOAD_LONG	DECIMAL	Long period duration. For example, load samples over last 15 minutes.	All except Windows operating systems	NULL
	CPU_USAGE_TOTAL	DECIMAL	Percentage of overall CPU usage of the machine.	All	Percent

Chapter 11. Explain routines

EXPLAIN_GET_MSGS

The EXPLAIN_GET_MSGS table function queries the EXPLAIN_DIAGNOSTIC and EXPLAIN_DIAGNOSTIC_DATA Explain tables, and returns formatted messages.

Syntax



The schema is the same as the Explain table schema.

Table function parameters

Any of the following input arguments can be null. If an argument is null, it is not used to limit the query.

explain-requester
An input argument of type VARCHAR(128) that specifies the authorization ID of the initiator of this Explain request. A null value excludes this parameter from the search condition of the query.

explain-time
An input argument of type TIMESTAMP that specifies the time of initiation for the Explain request. A null value excludes this parameter from the search condition of the query.

source-name
An input argument of type VARCHAR(128) that specifies the name of the package running when the dynamic statement was explained, or the name of the source file when the static SQL statement was explained. A null value excludes this parameter from the search condition of the query.

source-schema
An input argument of type VARCHAR(128) that specifies the schema, or qualifier, of the source of the Explain request. A null value excludes this parameter from the search condition of the query.

source-version
An input argument of type VARCHAR(64) that specifies the version of the source of the Explain request. A null value excludes this parameter from the search condition of the query.

explain-level
An input argument of type CHAR(1) that specifies the level of Explain information for which this row is relevant. A null value excludes this parameter from the search condition of the query.

stmtno
An input argument of type INTEGER that specifies the statement number

within the package to which this Explain information is related. A null value excludes this parameter from the search condition of the query.

sectno
An input argument of type INTEGER that specifies the section number within the package to which this Explain information is related. A null value excludes this parameter from the search condition of the query.

locale
An input argument of type VARCHAR(33) that specifies the locale of returned messages. If the specified locale is not installed on the DB2 server, the value is ignored.

Information returned

Table 94. Information returned by the EXPLAIN_GET_MSGS table function

Column name	Data type	Description
EXPLAIN_REQUESTER	VARCHAR(128)	Authorization ID of the initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	Time of initiation for the Explain request.
SOURCE_NAME	VARCHAR(128)	Name of the package running when the dynamic statement was explained, or the name of the source file when the static SQL statement was explained.
SOURCE_SCHEMA	VARCHAR(128)	Schema, or qualifier, of the source of the Explain request.
SOURCE_VERSION	VARCHAR(64)	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	Statement number within the package to which this Explain information is related.
SECTNO	INTEGER	Section number within the package to which this Explain information is related.
DIAGNOSTIC_ID	INTEGER	ID of the diagnostic for a particular instance of a statement in the EXPLAIN_STATEMENT table.
LOCALE	VARCHAR(33)	Locale of returned messages. This locale will not match the specified locale if the latter is not installed on the DB2 server.
MSG	VARCHAR(4096)	Formatted message text.

Examples

Request formatted English messages from the Explain tables in the default schema for requester SIMMEN that were generated in the last hour. Specify a source name of SQLC2E03.

```
SELECT MSG
  FROM TABLE(EXPLAIN_GET_MSGS(
    'SIMMEN',
    CAST(NULL AS TIMESTAMP),
    'SQLC2E03',
    CAST(NULL AS VARCHAR(128)),
    CAST(NULL AS VARCHAR(64)),
    CAST(NULL AS CHAR(1)),
    CAST(NULL AS INTEGER),
    CAST(NULL AS INTEGER),
    'en_US'))
  AS REGISTRYINFO
  WHERE EXPLAIN_TIME >= (CURRENT TIMESTAMP - 1 HOUR)
  ORDER BY DIAGNOSTIC_ID
```

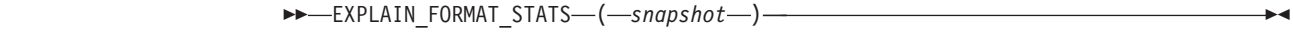
The following is an example of output from this query.

```
MSG
------------------------------------------------------------------------
EXP0012W  Invalid access request. The index "index1" could not be found.
  Line number "554", character number "20".
EXP0012W  Invalid access request. The index "index2" could not be found.
  Line number "573", character number "20".
EXP0015W  Invalid join request. Join refers to tables that are not in
  the same FROM clause. Line number "573", character number "20".
```

EXPLAIN_FORMAT_STATS

This new scalar function is used to display formatted statistics information which is parsed and extracted from explain snapshot captured for a given query. The data type of the result is CLOB(50M).

Syntax



The schema is SYSPROC.

Function parameters

snapshot
An input argument of type BLOB(10M) that is the explain snapshot captured for a given query. It is stored as snapshot column of explain table *EXPLAIN_STATEMENT*

Authorization

EXECUTE privilege on the EXPLAIN_FORMAT_STATS function.

Example

```
SELECT EXPLAIN_FORMAT_STATS(SNAPSHOT)
  FROM EXPLAIN_STATEMENT
  WHERE EXPLAIN_REQUESTER = 'DB2USER1' AND
      EXPLAIN_TIME = timestamp('2006-05-12-14.38.11.109432') AND
      SOURCE_NAME = 'SQLC2F0A' AND
      SOURCE_SCHEMA = 'NULLID' AND
      SOURCE_VERSION = '' AND
      EXPLAIN_LEVEL = 'O' AND
      STMTNO = 1 AND
      SECTNO = 201
```

The following is a sample output of this function:

```
Tablespace Context:
-------------------
Name:                                   USERSPACE1
Overhead:                               7.500000
Transfer Rate:                          0.060000
Prefetch Size:                          32
Extent Size:                            32
Type:                                   Database managed
Partition Group Name:                   NULLP
Buffer Pool Identifier:                 0

Base Table Statistics:
----------------------
Name:                                   T1
Schema:                                 DB2USER2
Number of Columns:                      3
Number of Pages with Rows:              1
Number of Pages:                        1
Number of Rows:                         5
Table Overflow Record Count:            0
Width of Rows:                          26
Time of Creation:                       2006-06-16-11.46.53.041085
Last Statistics Update:                 2006-06-26-12.23.44.814201
Statistics Type:                        Fabrication
Primary Tablespace:                     USERSPACE1
Tablespace for Indexes:                 USERSPACE1
Tablespace for Long Data:               NULLP
Number of Referenced Columns:           2
Number of Indexes:                      1
Volatile Table:                         No
Table Active Blocks:                    1
Number of Column Groups:                0
Number of Data Partitions:              1
Average Row Compression Ratio:          -9.000000
Percent Rows Compressed:                -9.000000
Average Compressed Row Size:            -9
Statistics Type:                        U

Column Information:
-------------------
Number:                                 1
Name:                                   C1
Statistics Available:                   Yes

Column Statistics:
------------------
Schema name of the column type:         SYSIBM
Name of column type:                    INTEGER
Maximum column length:                  4
Scale for decimal column:               0
Number of distinct column values:       4
Average column length:                  5
Number of most frequent values:         1
Number of quantiles:                    5
Second highest data value:              3
Second lowest data value:               2
Column sequence in partition key:       0
Average number of sub-elements:         -1
Average length of delimiters:           -1

Column Distribution Statistics:
-------------------------------
Frequency Statistics:
     Valcount   Value
     ---------- -----------
     2          1
```

```
Quantile Statistics:
     Valcount   Distcount   Value
     ---------- ----------- ----------
     0          1           1
     2          1           1
     3          2           2
     4          3           3
     5          4           4

Column Information:
--------------------
Number:                                  2
Name:                                    C2
Statistics Available:                    Yes

Column Statistics:
------------------
Schema name of the column type:          SYSIBM
Name of column type:                     INTEGER
Maximum column length:                   4
Scale for decimal column:                0
Number of distinct column values:        4
Average column length:                   5
Number of most frequent values:          1
Number of quantiles:                     5
Second highest data value:               3
Second lowest data value:                2
Column sequence in partition key:        0
Average number of sub-elements:          -1
Average length of delimiters:            -1

Column Distribution Statistics:
-------------------------------
Frequency Statistics:
      Valcount   Value
      ---------- -----------
      2          1

Quantile Statistics:
      Valcount   Distcount   Value
      ---------- ----------- -----------
      0          0           1
      2          0           1
      3          0           2
      4          0           4
      5          0           4

Indexes defined on the table:
-----------------------------
Name:                                    IDX_T1C1C2
Schema:                                  DB2USER2
Unique Rule:                             Duplicate index
Used in Operator:                        Yes
Page Fetch Pairs:                        Not Available
Number of Columns:                       2
Index Leaf Pages:                        1
Index Tree Levels:                       1
Index First Key Cardinality:             4
Index Full Key Cardinality:              4
Index Cluster Ratio:                     100
Index Cluster Factor:                    -1.000000
Time of Creation:                        2006-06-16-11.46.53.596717
Last Statistics Update:                  2006-06-26-12.23.44.814201
Index Sequential Pages:                  0
Index First 2 Keys Cardinality:          4
Index First 3 Keys Cardinality:          -1
```

```
Index First 4 Keys Cardinality:          -1
Index Avg Gap between Sequences:         0.000000
Fetch Avg Gap between Sequences:         -1.000000
Index Avg Sequential Pages:              0.000000
Fetch Avg Sequential Pages:              -1.000000
Index Avg Random Pages:                  1.000000
Fetch Avg Random Pages:                  -1.000000
Index RID Count:                         5
Index Deleted RID Count:                 0
Index Empty Leaf Pages:                  0
Avg Partition Cluster Ratio:             -1
Avg Partition Cluster Factor:            -1.000000
Data Partition Cluster Factor:           1.000000
Data Partition Page Fetch Pairs:         Not Available

Base Table Statistics:
----------------------
Name:                                    T2
Schema:                                  DB2USER2
Number of Columns:                       3
Number of Pages with Rows:               1
Number of Pages:                         1
Number of Rows:                          2
Table Overflow Record Count:             0
Width of Rows:                           26
Time of Creation:                        2006-06-16-11.46.53.398092
Last Statistics Update:                  2006-06-26-12.23.45.157028
Statistics Type:                         Synchronous
Primary Tablespace:                      USERSPACE1
Tablespace for Indexes:                  USERSPACE1
Tablespace for Long Data:                NULLP
Number of Referenced Columns:            2
Number of Indexes:                       1
Volatile Table:                          No
Table Active Blocks:                     -1
Number of Column Groups:                 0
Number of Data Partitions:               1

Column Information:
-------------------
Number:                                  1
Name:                                    C1
Statistics Available:                    Yes

Column Statistics:
------------------
Schema name of the column type:          SYSIBM
Name of column type:                     INTEGER
Maximum column length:                   4
Scale for decimal column:                0
Number of distinct column values:        2
Average column length:                   5
Number of most frequent values:          -1
Number of quantiles:                     2
Second highest data value:               2
Second lowest data value:                1
Column sequence in partition key:        0
Average number of sub-elements:          -1
Average length of delimiters:            -1

Column Distribution Statistics:
-------------------------------
Quantile Statistics:
      Valcount   Distcount   Value
      ---------- ----------- -----------
      1          1           1
      2          2           2
```

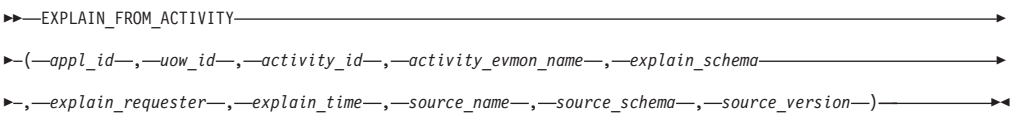
```
Column Information:
--------------------
Number:                                 2
Name:                                   C2
Statistics Available:                   Yes

Column Statistics:
------------------
Schema name of the column type:         SYSIBM
Name of column type:                    INTEGER
Maximum column length:                  4
Scale for decimal column:               0
Number of distinct column values:       2
Average column length:                  5
Number of most frequent values:         -1
Number of quantiles:                    2
Second highest data value:              2
Second lowest data value:               1
Column sequence in partition key:       0
Average number of sub-elements:         -1
Average length of delimiters:           -1

Column Distribution Statistics:
-------------------------------
Quantile Statistics:
      Valcount   Distcount   Value
      ---------- ---------- -----------
      1          0          1
      2          0          2

Indexes defined on the table:
-----------------------------
Name  :                                 IDX_T2C1
Schema:                                 DB2USER2
Unique Rule:                            Duplicate index
Used in Operator:                       No
Page Fetch Pairs:                       Not Available
Number of Columns:                      1
Index Leaf Pages:                       1
Index Tree Levels:                      1
Index First Key Cardinality:            2
Index Full Key Cardinality:             2
Index Cluster Ratio:                    100
Index Cluster Factor:                   -1.000000
Time of Creation:                       2006-06-16-11.46.53.857520
Last Statistics Update:                 2006-06-26-12.23.45.157028
Index Sequential Pages:                 0
Index First 2 Keys Cardinality:         -1
Index First 3 Keys Cardinality:         -1
Index First 4 Keys Cardinality:         -1
Index Avg Gap between Sequences:        0.000000
Fetch Avg Gap between Sequences:        -1.000000
Index Avg Sequential Pages:             0.000000
Fetch Avg Sequential Pages:             -1.000000
Index Avg Random Pages:                 1.000000
Fetch Avg Random Pages:                 -1.000000
Index RID Count:                        2
Index Deleted RID Count:                0
Index Empty Leaf Pages:                 0
Avg Partition Cluster Ratio:            -1
Avg Partition Cluster Factor:           -1.000000
Data Partition Cluster Factor:          1.000000
Data Partition Page Fetch Pairs:        Not Available
```

EXPLAIN_FROM_ACTIVITY procedure - Explain statement using activity event monitor information

The EXPLAIN_FROM_ACTIVITY procedure explains a specific execution of a statement using the contents of the section obtained from an activity event monitor.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

The Explain output is placed in the Explain tables for processing using any existing Explain tools (for example, db2exfmt). The Explain output contains, if available, both the access plan and section actuals (runtime statistics for operators in the access plan).



The schema is SYSPROC.

Authorization

All of the following privileges and authority are required:

- EXECUTE privilege on the EXPLAIN_FROM_ACTIVITY procedure
- INSERT privilege on the Explain tables in the specified schema
- SELECT privilege on the event monitor tables for the source activity event monitor

appl_id
An input argument of type VARCHAR(64) that uniquely identifies the application that issued the activity whose section is to be explained. If *appl_id* is null or an empty string, SQL2032N is returned.

uow_id
An input argument of type INTEGER specifying the unit of work identifier for the activity whose section is to be explained. Unit of work ID is only unique within a given application. If *uow_id* is null, SQL2032N is returned.

activity_id
An input argument of type INTEGER specifying the identifier of the activity whose section is to be explained. Activity ID is only unique within a unit of work. If *activity_id* is null, SQL2032N is returned.

activity_evmon_name
An input argument of VARCHAR(128) that specifies the name of a write to table activity event monitor containing the activity whose section is to be explained. If the event monitor does not exist or is not an activity event monitor, SQL0204N is returned. If the event monitor is not a write to table event monitor, SQL20502N is returned. If *activity_evmon_name* is not specified, SQL2032N is returned. If the caller does not have SELECT privilege on the activity event monitor tables, SQL0551N is returned.

explain_schema
An optional input or output argument of type VARCHAR(128) that specifies the schema containing the Explain tables where the explain information should be written. If an empty string or NULL is specified, a search is made for the explain tables under the session authorization ID and, following that, the SYSTOOLS schema. If the Explain tables cannot be found, SQL0219N is returned. If the caller does not have INSERT privilege on the Explain tables, SQL0551N is returned. On output, this parameter is set to the schema containing the Explain tables where the information was written.

explain_requester
An output argument of type VARCHAR(128) that contains the session authorization ID of the connection in which this routine was invoked.

explain_time
An output argument of type TIMESTAMP that contains the time of initiation for the Explain request.

source_name
An output argument of type VARCHAR(128) that contains the name of the package running when the statement was prepared or compiled.

source_schema
An output argument of type VARCHAR(128) that contains the schema, or qualifier, of the source of Explain request.

source_version
An output argument of type VARCHAR(64) that contains the version of the source of the Explain request.

Example

The following example assumes that you are mining the data collected in the activity event monitor over a period of time and using the following query, you notice a particularly expensive SQL statement in terms of CPU cost.

```
SELECT APPL_ID,
       UOW_ID,
       ACTIIVTY_ID,
       USER_CPU_TIME
FROM ACTIVITY_A
ORDER BY USER_CPU_TIME
```

The following example shows output from this query. The application with an ID of N2.DB2INST1.0B5A12222841 is using a large amount of CPU time.

```
   APPL_ID                  UOW_ID     ACTIVITY_ID    USER_CPU_TIME
   -----------------------  -------- -------------- ---------------
   *N2.DB2INST1.0B5A12222841   1              1       92782334234
   *N2.DB2INST1.0B5A12725841   2              7               326

     2 record(s) selected.
```

You can use the EXPLAIN_FROM_ACTIVITY procedure to investigate the access plan for this activity, to determine if the activity could benefit from tuning, for example, by adding an index.

```
CALL EXPLAIN_FROM_ACTIVITY( '*N2.DB2INST1.0B5A12222841', 1, 1, 'A', 'MYSCHEMA',
     ?, ?, ?, ?, ? )
```

Usage notes

In order to run Explain on the section of the activity, you must specify the COLLECT ACTIVITY DATA WITH SECTION clause when you enable collection of activity data, so that the section is collected with the activity information. If there is no section stored with the identified activity entry, SQL20501 is returned.

If section actuals were not collected for an activity, the section explain will succeed, but the Explain output will not include actuals information. Section actuals will not be collected in the following cases:

- The activity specified as input was captured using the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure. In this case, the value of the *partial_record* element in the activity logical group is 1.
- The activity event monitor ACTIVITY table is missing the SECTION_ACTUALS element.
- The section executed is a static section and it has not been rebound since applying DB2 Version 9.7 Fix Pack 1.
- Section actuals were not enabled for the section that was captured. Section actuals are enabled using the **section_actuals** database configuration parameter or for a specific application using the WLM_SET_CONN_ENV procedure. By default, section actuals are disabled.

Note: To verify that section actuals were collected for an activity, check whether the SECTION_ACTUALS element in the ACTIVITY table has a length greater than 0.

Note: The **section_actuals** setting specified by the WLM_SET_CONN_ENV procedure for an application takes effect immediately. Section actuals will be collected for the next statement issued by the application.

Note: In a partitioned database environment, section actuals will be collected only on partitions where activity data is collected. To collect actuals on all partitions, ensure the activity is collected using the COLLECT ACTIVITY DATA WITH DETAILS, SECTION ON ALL DATABASE PARTITIONS clause. If the user wishes to enable collection at all partitions for a particular application, they can include the <collect_act_partition> tag with a value of "ALL" in the second argument when calling the WLM_SET_CONN_ENV procedure.

If no activity can be found that corresponds to the *appl_id*, *uow_id*, and *activity_id* that you input, SQL20501 is returned. If multiple activities match, which may occur if an activity was collected multiple times during execution using the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure, the most recent entry for which a section was captured will be used for Explain.

The output parameters *explain_requester*, *explain_time*, *source_name*, *source_schema*, and *source_version* comprise the key used to look up the Explain information for the section in the Explain tables. Use these parameters with any existing Explain tools (for example, db2exfmt) to format the explain information retrieved from the section.

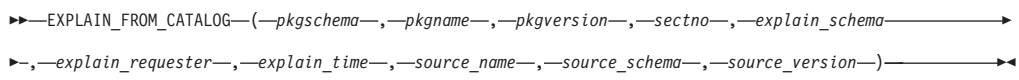
The EXPLAIN_FROM_ACTIVITY procedure does not issue a COMMIT statement after inserting into the Explain tables. It is the responsibility of the caller of the procedure to issue a COMMIT.

The following elements must be present in the ACTIVITYSTMT logical group: STMT_TEXT, ORIGINAL_STMT_TEXT, SECTION_ENV, EXECUTABLE_ID, APPL_ID, ACTIVITY_ID, UOW_ID. If any of these elements are missing, the stored procedure returns SQL206.

EXPLAIN_FROM_CATALOG procedure - Explain a statement using section information from catalogs

The EXPLAIN_FROM_CATALOG procedure explains a statement using the contents of the section obtained from the catalogs. The Explain output is placed in the Explain tables for processing using any existing explain tools (for example, db2exfmt).

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.



The schema is SYSPROC.

Authorization

All of the following privileges and authority are required:

- EXECUTE privilege on the EXPLAIN_FROM_CATALOG procedure
- INSERT privilege on the Explain tables in the specified schema
- EXPLAIN authority

pkgschema
An input argument of type VARCHAR(128) specifying the schema of the package containing the section to be explained. If *pkgschema* is null or an empty string, SQL2032N is returned.

pkgname
An input argument of type VARCHAR(128) specifying the package containing the section to be explained. If *pkgname* is null or an empty string, SQL2032N is returned.

pkgversion
An input argument of type VARCHAR(64) specifying the version identifier for the package containing the section to be explained. Specify an empty string if the package has no version (a blank ' ' character if VARCHAR2 compatibility mode is enabled). If *pkgversion* is null, SQL2032N is returned.

sectno
An input argument of type SMALLINT specifying the section to be explained. If *sectno* is null, SQL2032N is returned.

explain_schema
An optional input or output argument of type VARCHAR(128) that specifies the schema containing the Explain tables where the explain information should be written. If an empty string or NULL is specified, a search is made for the explain tables under the session authorization ID and, following that, the

SYSTOOLS schema. If the Explain tables cannot be found, SQL0219N is returned. If the caller does not have INSERT privilege on the Explain tables, SQL0551N is returned. On output, this parameter is set to the schema containing the Explain tables where the information was written.

explain_requester
An output argument of type VARCHAR(128) that contains the session authorization ID of the connection in which this routine was invoked.

explain_time
An output argument of type TIMESTAMP that contains the time of initiation for the Explain request.

source_name
An output argument of type VARCHAR(128) that contains the name of the package running when the statement was prepared or compiled.

source_schema
An output argument of type VARCHAR(128) that contains the schema, or qualifier, of the source of Explain request.

source_version
An output argument of type VARCHAR(64) that contains the version of the source of the Explain request.

Example

The following example demonstrates how to explain a static statement that was compiled and exists in the catalogs. First, you can identify the section by selecting from the SYSCAT.STATEMENTS catalog view, for example:

```
SELECT pkgschema,
       pkgname,
       version,
       Sectno
FROM SYSCAT.STATEMENTS
WHERE TEXT = 'select count(*) from syscat.tables'
```

This query returns the following sample output:

```
PKGSCHEMA  PKGNAME              PKGVERSION           SECTNO
---------- -------------------- -------------------- ------
NULLID     SQLE2G0S                                  1
NULLID     SQLE2G0S             VERSION1             1

     2 record(s) selected.
```

Then pass the *pkgschema*, *pkgname*, *pkgversion* and *sectno* identification information into the EXPLAIN_FROM_CATALOG procedure, for example:

```
CALL EXPLAIN_FROM_CATALOG( 'NULLID', 'SQLE2G0S', '', 1, 'MYSCHEMA', ?, ?, ?, ?, ? )
```

Usage notes

If no section can be found corresponding to the input parameters, SQL20501 is returned.

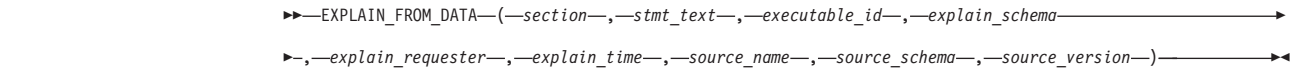
The output parameters *explain_requester*, *explain_time*, *source_name*, *source_schema*, *source_version* comprise the key used to look up the Explain information for the section in the Explain tables. Use these parameters with any existing Explain tools (for example, db2exfmt) to format the explain information retrieved from the section.

The procedure does not issue a COMMIT statement after inserting into the Explain tables. It is the responsibility of the caller of the procedure to issue a COMMIT.

EXPLAIN_FROM_DATA procedure - Explain a statement using the input section

The EXPLAIN_FROM_DATA procedure explains a statement using the contents of the input section. The Explain output is placed in the Explain tables for processing using any existing Explain tools (for example, db2exfmt).

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.



The schema is SYSPROC.

Authorization

All of the following privileges and authority are required:

- EXECUTE privilege on the EXPLAIN_FROM_DATA procedure
- INSERT privilege on the Explain tables in the specified schema

section
An input argument of type BLOB(134M) that contains the section to be explained. You can obtain the section from various sources, including event monitor tables and the catalog tables. If the input section is not a valid section, SQL20503N is returned.

stmt_text
An optional input argument of type CLOB(2M) that contains the text of the statement corresponding to the input section. If *stmt_text* is NULL, the formatted Explain output will not contain any statement text.

executable_id
An optional input argument of type VARCHAR(32) FOR BIT DATA that contains the executable ID used to identify the section. If *executable_id* is NULL, the formatted explain output will not contain an executable ID.

explain_schema
An optional input or output argument of type VARCHAR(128) that specifies the schema containing the Explain tables where the explain information should be written. If an empty string or NULL is specified, a search is made for the explain tables under the session authorization ID and, following that, the SYSTOOLS schema. If the Explain tables cannot be found, SQL0219N is returned. If the caller does not have INSERT privilege on the Explain tables, SQL0551N is returned. On output, this parameter is set to the schema containing the Explain tables where the information was written.

explain_requester
An output argument of type VARCHAR(128) that contains the session authorization ID of the connection in which this routine was invoked.

explain_time
An output argument of type TIMESTAMP that contains the time of initiation for the Explain request.

source_name
An output argument of type VARCHAR(128) that contains the name of the package running when the statement was prepared or compiled.

source_schema
An output argument of type VARCHAR(128) that contains the schema, or qualifier, of the source of Explain request.

source_version
An output argument of type VARCHAR(64) that contains the version of the source of the Explain request.

Example

Assume you have captured a number of statements using the package cache event monitor and extracted the event monitor data (using the EVMON_FORMAT_UE_TO_TABLE stored procedure) to a table named PKGCACHE. Looking at the data in the table, you identify a particularly expensive statement which has executable id x'0100000000000000070000000000000000000000000020020081126190410369 8'.

You issue the EXPLAIN_FROM_DATA procedure to understand the access plan for this statement, passing as input the section from the entry in the PKGCACHE table. You place the Explain output in the explain tables in the MYSCHEMA schema.

```
SET SERVEROUTPUT ON;

BEGIN
  DECLARE EXECUTABLE_ID VARCHAR(32) FOR BIT DATA; --
  DECLARE SECTION BLOB(134M); --
  DECLARE STMT_TEXT CLOB(2M); --
  DECLARE EXPLAIN_SCHEMA VARCHAR(128); --

  DECLARE EXPLAIN_REQUESTER VARCHAR(128); --
  DECLARE EXPLAIN_TIME TIMESTAMP; --
  DECLARE SOURCE_NAME VARCHAR(128); --
  DECLARE SOURCE_SCHEMA VARCHAR(128); --
  DECLARE SOURCE_VERSION VARCHAR(128); --

  SET EXPLAIN_SCHEMA = 'MYSCHEMA'; --

  SELECT P.SECTION, P.STMT_TEXT, P.EXECUTABLE_ID INTO
         SECTION, STMT_TEXT, EXECUTABLE_ID
  FROM PKGCACHE WHERE EXECUTABLE_ID =
   x'0100000000000000070000000000000000000000000020020081126190410369 8'; --

  CALL EXPLAIN_FROM_DATA( SECTION,
                          STMT_TEXT,
                          EXECUTABLE_ID,
                          EXPLAIN_SCHEMA,
                          EXPLAIN_REQUESTER,
                          EXPLAIN_TIME,
                          SOURCE_NAME,
                          SOURCE_SCHEMA,
                          SOURCE_VERSION ); --

  CALL DBMS_OUTPUT.PUT( 'EXPLAIN_REQUESTER = ' ); --
  CALL DBMS_OUTPUT.PUT_LINE( EXPLAIN_REQUESTER ); --
  CALL DBMS_OUTPUT.PUT( 'EXPLAIN_TIME = ' ); --
```

```
  CALL DBMS_OUTPUT.PUT_LINE( EXPLAIN_TIME ); --
  CALL DBMS_OUTPUT.PUT( 'SOURCE_NAME = ' ); --
  CALL DBMS_OUTPUT.PUT_LINE( SOURCE_NAME ); --
  CALL DBMS_OUTPUT.PUT( 'SOURCE_SCHEMA = ' ); --
  CALL DBMS_OUTPUT.PUT_LINE( SOURCE_SCHEMA ); --
  CALL DBMS_OUTPUT.PUT( 'SOURCE_VERSION = ' ); --
  CALL DBMS_OUTPUT.PUT_LINE( SOURCE_VERSION ); --
END;

SET SERVEROUTPUT OFF;
```

Usage notes

The input section can be obtained from a number of different sources:

- Activity event monitor
- Package cache event monitor
- Catalog tables
- Any user table or input source that has made a copy of the section from one of the locations, above.

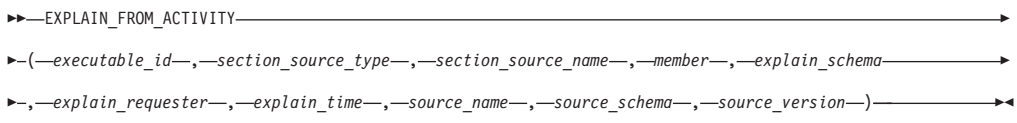
The output parameters *explain_requester*, *explain_time*, *source_name*, *source_schema*, *source_version* comprise the key used to look up the Explain information for the section in the Explain tables. Use these parameters with any existing Explain tools (for example, db2exfmt) to format the explain information retrieved from the section.

The procedure does not issue a COMMIT after inserting into the Explain tables. It is the responsibility of the caller of the procedure to issue a COMMIT.

EXPLAIN_FROM_SECTION procedure - Explain a statement using package cache or package cache event monitor information

The EXPLAIN_FROM_SECTION procedure explains a statement using the contents of the section obtained from the package cache or from the package cache event monitor. The Explain output is placed in the Explain tables for processing using any existing explain tools (for example, db2exfmt).

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.



The schema is SYSPROC.

Authorization

All of the following privileges and authority are required:

- EXECUTE privilege on the EXPLAIN_FROM_SECTION procedure
- INSERT privilege on the Explain tables in the specified schema

- SELECT privilege on the package cache event monitor table, if the section source name identifies a package cache event monitor

executable_id
An input argument of type VARCHAR(32) FOR BIT DATA that uniquely identifies a section to be explained. If this argument is null or an empty string, SQL2032 is returned.

section_source_type
An input argument of type CHAR(1) that specifies the source of the section to be explained. Valid values are:

- M - Section is obtained from the in-memory package cache
- P - Section is obtained from a package cache event monitor

For static SQL, if the *section_source_type* is M and the section cannot be located in the package cache, the catalog tables are searched for the section.

section_source_name
An input argument of VARCHAR(128) that specifies the name of a package cache event monitor if the *section_source_type* is P. If the *section_source_type* is M, the name of a package cache event monitor can be optionally specified. The event monitor is searched for the section if the section cannot be found in the package cache (for example, if the section was flushed from the package cache before the EXPLAIN_FROM_SECTION stored procedure was invoked). If the source input event monitor is not a package cache event monitor created with the COLLECT DETAILED DATA option, SQL0204N is returned. If the caller does not have SELECT privilege on the package cache event monitor table, SQL0551N is returned.

member
An input argument of type INTEGER that specifies the member where the section to be explained resides in memory if the *section_source_type* is M. If -1 is specified, the procedure searches for the section on the current coordinator member and the section compilation member. This argument is ignored if the *section_source_type* is anything other than M.

explain_schema
An optional input or output argument of type VARCHAR(128) that specifies the schema containing the Explain tables where the explain information should be written. If an empty string or NULL is specified, a search is made for the explain tables under the session authorization ID and, following that, the SYSTOOLS schema. If the Explain tables cannot be found, SQL0219N is returned. If the caller does not have INSERT privilege on the Explain tables, SQL0551N is returned. On output, this parameter is set to the schema containing the Explain tables where the information was written.

explain_requester
An output argument of type VARCHAR(128) that contains the session authorization ID of the connection in which this routine was invoked.

explain_time
An output argument of type TIMESTAMP that contains the time of initiation for the Explain request.

source_name
An output argument of type VARCHAR(128) that contains the name of the package running when the statement was prepared or compiled.

source_schema
An output argument of type VARCHAR(128) that contains the schema, or qualifier, of the source of Explain request.

source_version
An output argument of type VARCHAR(64) that contains the version of the source of the Explain request.

Example

This example shows how to identify and analyze a particularly expensive statement in the package cache by looking at the monitoring metrics available per section. First, issue a query similar to the following SELECT statement to determine the CPU time usage of sections.

```
SELECT executable_id,
       Total_cpu_time,
       Varchar(stmt_text, 100) as stmt_text
FROM TABLE(MON_GET_PKG_CACHE_STMT (NULL, NULL,NULL, -1)) AS T
```

The following sample shows output from this query.

```
EXECUTABLE_ID                                 TOTAL_CPU_TIME ...
--------------                                -------------- ...
x'010000000000000012...20020081126190410369 8' 91875622       ...
x'010000000000000007...20020081126190410323 8'      300       ...

     2 record(s) selected.
```

The following sample continues the output from this query.

```
...STMT_TEXT
...----------------------
...SELECT * FROM SYSCAT.TABLES
...INSERT INTO T1 VALUES(123)

     2 record(s) selected.
```

To examine the access plan for the expensive SELECT statement, pass its *executable_id* to the EXPLAIN_FROM_SECTION procedure. Place the output in the Explain tables in the MYSCHEMA schema.

```
CALL EXPLAIN_FROM_SECTION
     ( x'0100000000000000120000000000000000000000002002008112619041036 98',
      'M', NULL, 0, 'MYSCHEMA', ?, ?, ?, ?, ? )
```

Usage notes

If the section corresponding to the input executable ID cannot be found, SQL20501 is returned. The input *executable_id* can be obtained from the following sources

- Activity event monitor
- Package cache event monitor
- MON_GET_ACTIVITY_DETAILS table function
- MON_GET_PKG_CACHE_STMT table function
- WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 table function
- WLM_GET_SERVICE_CLASS_AGENTS_V97 table function
- MON_GET_PKG_CACHE_STMT_DETAILS table function
- MON_GET_APP_LOCK_WAITS table function

The output parameters *explain_requester*, *explain_time*, *source_name*, *source_schema*, *source_version* comprise the key used to look up the information for the section in the Explain tables. Use these parameters with any existing Explain tools (for example, db2exfmt) to format the explain information retrieved from the section.

The procedure does not issue a COMMIT statement after inserting into the Explain tables. It is the responsibility of the caller of the procedure to issue a COMMIT.

Chapter 12. Monitor routines and views

Monitor table functions and views are routines with names that begin with "MON", such as MON_GET_SERVICE_SUBCLASS. These table functions and views provide access to monitor elements that are available from the monitoring infrastructure introduced in DB2 Version 9.7. Certain other routines, such as snapshot functions, also return monitoring information.

The "MON" routines are strategically important, therefore the names of these routines will not change in future releases. However, they will have new output columns added when enhancements are made in future releases. Therefore, when you issue a query to retrieve information using a system-defined routine or view, do not use a statement of the form SELECT * .... Instead, name the result columns in the SELECT statement. This gives the application control over the number of result columns and the sequence in which they are returned.

Monitor (MON) table functions

All table functions include a common set of monitor elements. These elements provide information about a diverse set of system performance indicators that can affect application response time. You can also obtain monitor data for a subset of the workload you are interested in.

Some monitor table functions report on various aspects of the overall system workload, for example:

- MON_GET_CONNECTION and MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS and MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_UNIT_OF_WORK and MON_GET_UNIT_OF_WORK_DETAILS
- MON_GET_WORKLOAD and MON_GET_WORKLOAD_DETAILS

These table functions have two versions, one of which has a _DETAILS suffix. The version without the _DETAILS suffix provides a relational SQL interface that returns the most commonly used data. The version with the _DETAILS suffix provides XML-based access to the monitor data, and returns a more comprehensive data set.

Other table functions return data for a specific type of data object, for example:

- MON_GET_APPL_LOCKWAIT
- MON_GET_BUFFERPOOL
- MON_GET_CONTAINER
- MON_GET_INDEX
- MON_GET_LOCKS
- MON_GET_TABLE
- MON_GET_TABLESPACE
- MON_GET_PKG_CACHE_STMT

Use these table functions to investigate performance issues associated with a particular data object.

The following table functions, which return data about fast communication manager (FCM), have been added with the Version 9.7 Fix Pack 2 release:

- MON_GET_FCM
- MON_GET_FCM_CONNECTION_LIST

Other table functions are useful for examining details of individual activities and statements:

- MON_GET_ACTIVITY_DETAILS returns details for a specific activity currently running on the system; these details include general activity information (like statement text) and a set of metrics.

In addition, this table function serves a progress monitoring role:

- MON_GET_EXTENT_MOVEMENT_STATUS returns the status of the extent movement operation.

The table functions that begin with MON_FORMAT_ return information in an easy-to-read row-based format. The MON_FORMAT_LOCK_NAME takes the internal binary name of a lock and returns detailed information about the lock. The table functions that begin with MON_FORMAT_XML_ take as input an XML metrics document returned by one of the MON_GET_*_DETAILS table functions (or from the output of statistics, activity, unit of work, or package cache event monitors) and returns formatted row-based output.

- MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW returns formatted row-based output on component times.
- MON_FORMAT_XML_METRICS_BY_ROW returns formatted row-based output for all metrics.
- MON_FORMAT_XML_TIMES_BY_ROW returns formatted row-based output on the combined hierarchy of wait and processing times.
- MON_FORMAT_XML_WAIT_TIMES_BY_ROW table function returns formatted row-based output on wait times.

Characteristics of monitor (MON) table functions

- The metrics returned by the monitoring table functions are never reset. They start at 0 when the database is activated and continue to accumulate until the database is deactivated.
- With most table functions, you can choose to receive data for a single object (for example, service class "A") or for all objects.
- As with most table functions, when using these table functions in a partitioned database environment, you can choose to receive data for a single partition or for all partitions. If you choose to receive data for all partitions, the table functions return one row for each partition. You can add the values across partitions to obtain the value of a monitor element across partitions.

Monitor (MON) views

The monitor views return metrics on various database activities, for example:

- MON_CURRENT_SQL returns metrics for all activities that were submitted on all members of the database and have not yet been completed, including a point-in-time view of currently executing SQL statements.
- MON_DB_SUMMARY returns metrics aggregated over all service classes.
- MON_LOCKWAITS returns information about agents working on behalf of applications that are waiting to obtain locks in the currently connected database.

- MON_SERVICE_SUBCLASS_SUMMARY returns metrics for all service subclasses, showing work executed per service class.
- MON_CURRENT_UOW returns metrics for all units of work.
- MON_WORKLOAD_SUMMARY returns metrics for all workloads, showing incoming work per workload.

Event monitor (EVMON) routines

The DB2 Version 9.7 release also introduces two new routines whose purpose is somewhat different than the other "MON" table functions. These routines extract and format data from event monitors that write events to an unformatted event table. The LOCKING and UNIT OF WORK event monitor types use unformatted event tables. The routines names are as follows:

- EVMON_FORMAT_UE_TO_XML table function
- EVMON_FORMAT_UE_TO_TABLES procedure

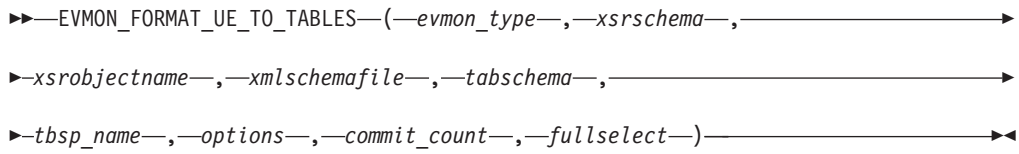
These routines allow you to access event monitor data, either through an XML document, by using the EVMON_FORMAT_UE_TO_XML table function; or through relational tables, by using the EVMON_FORMAT_UE_TO_TABLES procedure.

EVMON_FORMAT_UE_TO_TABLES procedure - move an XML document to relational tables

The EVMON_FORMAT_UE_TO_TABLES procedure retrieves data stored in an unformatted event (UE) table produced by an event monitor and converts it into a set of relational tables.

The process of creating relational tables takes place in two steps. First the data in the UE table is converted to XML format, using the EVMON_FORMAT_UE_TO_XML table function. This table function is run for you automatically as part of running the EVMON_FORMAT_UE_TO_TABLES procedure. Next, the XML document that contains the event monitor data is turned into relational tables using XML decomposition.

Syntax



The schema is SYSPROC.

Table function parameters

evmon_type
An input parameter of type VARCHAR(128) that represents the type of data stored in the unformatted event table. The possible values are as follows:

LOCKING
Data stored in the unformatted event table is from a locking event monitor.

PKGCACHE
Data stored in the unformatted event table is from a PACKAGE CACHE event monitor.

UOW Data stored in the unformatted event table is from a UOW event monitor.

xsrschema
An input parameter of type VARCHAR (128) that specifies the first-part of the name of the XSR object that describes how data from the UE file corresponds to columns in tables. The second-part of the XSR object name is derived from the *xsrobjectname* parameter. The complete XSR object name is defined as *xsrschema.xsrobjectname*. If this value is NULL, then the authorization ID of the current session user is used.

xsrobjectname
An input parameter of type VARCHAR (128) that specifies the second-part of the name of the XSR object that describes how data from the UE file corresponds to columns in tables. The first-part of the XSR object name is derived from the *xsrschema* parameter. The complete XSR object name is defined as *xsrschema.xsrobjectname* and is unique among all objects in the XSR. If this value is NULL then the *xsrobjectname* is derived as follows: `EVMON_<evmon_type>_SCHEMA_<SQL release level>`. For example, a locking event monitor in DB2 Version 9.7 would have a derived *xsrname* of `EVMON_LOCKING_SCHEMA_SQL09070`.

The XSR object is a copy of the XML schema file that describes the output of the event monitor. It is stored in the XML schema repository (XSR), and defines the relationship between the elements of the interim XML document produced by the first stage of EVMON_FORMAT_UE_TO_TABLES processing, and the tables and columns the procedure ultimately produces. The XSR object is also used to manage the mutual dependency between any tables that have been created and the XML schema from which those tables are derived. If the XSR object is dropped, or if any of the tables produced by the procedure are dropped or the columns altered, the dependency between the two is said to be broken. If EVMON_FORMAT_UE_TO_TABLES (or the EVMON_FORMAT_UE_TO_XML table function) has not yet been run against the UE file for a specific type of event monitor, the XSR object that describes the event monitor output will not yet exist. In this case, the XML schema file for the event monitor is used to create and register an XSR object in the system catalog tables.

xmlschemafile
An input parameter of type VARCHAR (1024) that is a fully qualified path to the XML schema document on disk that describes the output produced by the event monitor. The XML schema document elements are annotated with information that maps XML elements and attributes to the relational tables and their columns.

This parameter is used register an XSR object. If there is no XSR object registered and enabled for the type of event monitor specified in *evmon_type*, then an XSR object is registered as follows:

- If *xmlschemafile* is NULL, then the procedure uses the XML schema file on disk that corresponds to value specified for *evmon_type*, as follows:

  LOCKING
  `sqllib/misc/DB2EvmonLocking.xsd`

  PKGCACHE
  `sqllib/misc/DB2EvmonPkgCache.xsd`

UOW `sqllib/misc/DB2EvmonUOW.xsd`

- If you specify the name of an XML schema file, then that file is used to register and enable the XSR object for decomposition.
- If you specify values for the *xsrschema* and *xsrobjectname* parameters, then XSR object is created with these names. Otherwise, the XSR object is named as using the defaults previously described for *xsrobjectname*.

Important: If an XSR object has previously been registered and is enabled for decomposition, this parameter is ignored. If you want to register an XSR object using a different XML schema file, you must first drop the existing XSR object.

tabschema
An input parameter of type VARCHAR (128) that represents the SQL schema name where the event monitor relational tables are created. If this value is NULL, then the authorization ID of the current session user is used. The SQL schema under which the tables are created is determined as follows:

- If `<db2-xdb:SQLSchema>` is specified, use this schema;
- If `<db2-xdb:defaultSchema>` is specified, use this schema;
- If neither of these values is specified, use the value from the *sqlschema* input parameter.

Note: When an XML schema is registered for decomposition, the XSR schema repository creates a dependency between each table referenced in the schema and the XSR object that corresponds to this schema. Which means the XSR object name is linked to a unique set of relational tables in the database. If you reference an existing XSR object, its data is always decomposed and inserted into the tables to which the XSR object was linked.

tbsp_name
An input parameter of type VARCHAR(128) that indicates the table space where the relational tables are created. The default value for this parameter is NULL. The table space name specified on the CREATE TABLE statement within the XML schema file takes precedence over this input parameter.

options
An input parameter of type VARCHAR(1024) which represents a list of keyword options supported by this table function. Each option must be delimited using a semicolon (;) character. The possible values are:

RECREATE_FORCE
Indicates that the relational tables are dropped and re-created before decomposition.

RECREATE_ONERROR
Indicates that the relational tables are dropped and recreated in the following situations:

1. If the XSR object is not registered, but the tables exist.
2. On the first failed decomposition attempt. Subsequent failures are returned, and no attempts are made to re-create the tables.

If an error occurs, for example, a table space full error or an authorization error, the procedure does not filter the SQLCODE returned by the decomposition procedure. The procedure treats all negative SQLCODES equally and tries to re-create the tables.

commit_count
An input parameter of type INTEGER. The possible values are as follows:

-1 Commit after every 100 successful documents decomposed.-1 is the default value.

0 Never commit.

n Commit after every *n* documents successfully decomposed.

fullselect
An input parameter of type CLOB(2M) that represents the fullselect statement from an unformatted event table. The fullselect statement is a query that conforms to the rules of the SELECT statement. The query must follow the following rules:

- The query must use the "*" clause or specify all the columns of the unformatted event table. Otherwise an error is returned. The columns must be specified in the same order as returned by the DESCRIBE statement of the unformatted event table.
- The query must select only from an unformatted event table.
- The WHERE clause can use any of the non-LOB columns of the unformatted event table to filter out events.

Authorization

EXECUTE privilege on the EVMON_FORMAT_UE_TO_TABLES stored procedure.

SELECT privilege on the unformatted event table, if you did not create it.

CREATE privilege to create the relational tables in the specified SQL schema.

INSERT privilege to insert into the relational tables, if you did not create them.

All privileges required by the XDB_DECOMP_XMP_FROM_QUERY procedure.

Usage notes

Table creation

In order for decomposition to occur, a set of relational tables must exist. The EVMON_FORMAT_UE_TO_TABLES procedure creates the relational tables automatically, as follows:

- The procedure parses the event monitor XML schema file to find the `<db2-mon:createStmt>` elements. Each element contains a complete CREATE TABLE statement.
- The procedure extracts and runs the CREATE TABLE statements.

The `<db2-mon:createStmt>` is a child element of the existing `<db2-xdb:table>` element. Only the EVMON_FORMAT_UE_TO_TABLES procedure recognizes and uses this element. All other procedures that parse the XML schema file, such as the XSR objects, ignore this element.

Do not qualify the table name within the `<db2-mon:createStmt>`.

XML schema files from release to release

The default XML schema files provided by each event monitor always reflects the XML schema for the current release. So, when you run EVMON_FORMAT_UE_TO_TABLES (or EVMON_FORMAT_UE_TO_XML), the

output reflects the monitor elements defined for that event monitor in that release. The next section describes what happens if the schema files for the event monitors happen to change over time. Understanding the impact of these changes is important if you create tables using the EVMON_FORMAT_UE_TO_TABLES procedure, and then apply a fix pack or upgrade to a new release.

Impact of schema updates on tables produced by EVMON_FORMAT_UE_TO_TABLES

New monitor elements are likely to be added to event monitors in future fix packs or releases. These new monitor elements might result in new columns or even new tables being produced by the EVMON_FORMAT_UE_TO_TABLES procedure. However, if you already have tables that were created by this procedure before a fix pack was applied, or before upgrading to a new release, you need to do the following to have the new relational columns or tables created:

For fix pack updates

If relational tables produced by EVMON_FORMAT_UE_TO_TABLES before the installation of the latest fix pack still exist, you must force the creation of a new set of tables based on the new schema shipped in the fix pack if you want to see the new monitor elements in relational format.

To force the EVMON_FORMAT_UE_TO_TABLES procedure to use the new schema shipped in the fix pack and create new tables, perform the following steps:

1. Break the dependency between the currently registered version of the XML schema (see the note under the *tabschema* parameter of the EVMON_FORMAT_UE_TO_TABLES procedure for more information about schema registration) and the existing tables by performing one of the following actions:
   - Drop one of the existing tables that were produced by EVMON_FORMAT_UE_TO_TABLES
   - Drop the registered XML schema object associated with the existing tables using the DROP XSROBJECT statement. For example, to drop the registered XML schema object associated with the tables produced by EVMON_FORMAT_UE_TO_TABLES for the locking event monitor for DB2 V9.7, use the following command: `DROP XSROBJECT EVMON_LOCKING_SCHEMA_SQL09070.`
   - Alter any existing column that corresponds to an annotated monitor element in the currently registered XML schema object.
2. Run the EVMON_FORMAT_UE_TO_TABLES procedure, using the FORCE option. This option causes the old tables to be dropped, and a new set of tables to be produced. If you omit this option, a SQL0601N error is returned.

This process is illustrated in "Example 5: Picking up new elements in a fix pack update" on page 362.

If you do not perform the preceding steps, existing tables are updated based on the previously registered schema file. Any new columns or tables that might have been added in the fix pack are not reflected in the output of the EVMON_FORMAT_UE_TO_TABLES procedure.

For release upgrades

Unless you specify otherwise, the default version of the XML schema file for the current release is used when you call the EVMON_FORMAT_UE_TO_TABLES procedure. So, if you upgrade to a

new release of the DB2 product, then, by default, the new version of the schema file is used when you run the procedure.

If tables from the previous release do not exist, EVMON_FORMAT_UE_TO_TABLES produces tables using the most recent schema. However, if tables from the previous release exist, you must use the FORCE or RECREATE_ONERROR options to cause the old tables to be replaced by new ones. Otherwise, a SQL0601N error is returned. "Example 6: Picking up new elements in a release update" on page 362 shows an example of recreating the tables using the default schema for a new release.

Alternatively, you can continue to use the existing tables, without adding any new columns or tables that might have been introduced in the latest release. To have the existing tables updated, you must specify the name of the registered XML schema file that was used to create the tables for the *xsrobjectname* parameter of the EVMON_FORMAT_UE_TO_TABLES procedure. "Example 7: Using the previous relational tables on a release update" on page 362 shows an example of using the schema from a previous release.

Note: You cannot pick up any new columns or tables introduced in fix packs or in new releases while retaining the data that was previously in the relational tables produced by EVMON_FORMAT_UE_TO_TABLES. Picking up any new columns requires the tables to be recreated.

Examples

Example 1: Using default parameters

A user named Paul calls the procedure using the default parameters and requires all events that are part of the service class STUDENTS to be inserted into the relational tables.

```
EVMON_FORMAT_UE_TO_TABLES (
  'UOW', NULL, NULL, NULL, NULL, NULL, NULL, -1,
  'SELECT * FROM UOWUE
     WHERE service_subclass_name = 'STUDENTS'
     ORDER BY event_id, event_timestamp')
```

The results of the call are as follows:

1. The procedure parses the `DB2EvmonUOW.xsd` file, which is the default XML schema file, to identify the set of relational tables to create.
2. The relational tables are created under SQL schema Paul.
3. The XML schema is registered with an XSR object name of `PAUL.EVMON_UOW_SCHEMA_SQL09070`
4. XSR object is enabled for decomposition.

5. Data is decomposed and inserted into the tables under SQL schema Paul.

Example 2: Attempting to use tables under a different schema

In a continuation of the previous example, a user named Dave calls the stored procedure, setting the *tabschema* parameter to Paul.

```
EVMON_FORMAT_UE_TO_TABLES (
  'UOW', NULL, NULL, NULL, 'Paul', NULL, NULL, -1,
  'SELECT * FROM UOWTBLE
     ORDER BY event_timestamp')
```

The results of the call are as follows:

1. The procedure parses the `DB2EvmonUOW.xsd` file, which is the default XML schema file, to identify the set of relational tables to create.
2. The procedure attempts to create the tables under schema Paul. However, an error is returned because the relational tables currently exist under the SQL schema PAUL. Previously existing tables cannot be used when a new XSR object is being registered.

Example 3: Attempting to use tables under a different schema

In a continuation of the previous example, a user named Greg calls the stored procedure setting the input parameter *xsrschema* to Paul.

```
EVMON_FORMAT_UE_TO_TABLES (
  'UOW', 'Paul', NULL, NULL, NULL, NULL, NULL, -1,
  'SELECT * FROM UOWTBL
     ORDER BY event_timestamp')
```

The results of the call are as follows:

1. The XSR object `Paul.EVMON_UOW_SCHEMA_SQL09070`, which exists, is enabled for decomposition.
2. If Greg has INSERT privileges on the tables, then data is decomposed and inserted into the relational tables under SQL schema Paul. The existing XSR object `Paul.EVMON_UOW_SCHEMA_SQL09070` is used, so the SQL schema for the relational tables is obtained from the XSR object, instead of being provided as an input parameter to the procedure.

Example 4: Using the RECREATE_FORCE option

In a continuation of the previous example, Paul wants to re-create the tables again, but in table space MYSPACE. Paul calls the procedure with the RECREATE_FORCE option and the *tbsp_name* parameter.

```
EVMON_FORMAT_UE_TO_TABLES (
  'UOW', NULL, NULL, NULL, NULL, 'MYSPACE', 'RECREATE_FORCE', -1,
  'SELECT * FROM UOWTBL
     ORDER BY event_timestamp')
```

The results of the call are as follows:

1. The XSR object `Paul.EVMON_UOW_SCHEMA_SQL09070`, which exists, is enabled for decomposition.
2. The RECREATE_FORCE option is set.
3. The XML schema file is retrieved from the schema repository and parsed to identify the set of relational files.
4. The current tables are dropped and created again in the MYSPACE table space.
5. Data is decomposed and inserted into the new tables.

Example 5: Picking up new elements in a fix pack update

A new XML element called "db2EventNew" has been added to the XML schema file of the locking event monitor in the latest fix pack. Paul wants to pick up the new element to use in the decomposition of an XML file. To do so, he follows the following steps:

1. Paul drops the XSR object created in the original release:
   ```
   DROP XSROBJECT EVMON_LOCKING_SCHEMA_SQL09070
   ```
2. He calls the procedure with the RECREATE_ONERROR option.
   ```
   EVMON_FORMAT_UE_TO_TABLES (
     'LOCKING', NULL, NULL, NULL, NULL, NULL, 'RECREATE_ONERROR', -1,
     'SELECT * FROM LOCK
        ORDER BY event_timestamp')
   ```
   The results of the call are as follows:
   a. The XSR object does not exist, so the default `DB2EvmonLocking.xsd` schema file is parsed to identify the set of relational tables.
   b. As the RECREATE_ONERROR option was specified, the existing tables are dropped and re-created.

Example 6: Picking up new elements in a release update

Paul is upgrading to a new DB2 release and wants to pick up the new changes in the event monitor XML schema file. Paul calls the procedure with the RECREATE_ONERROR option.

```
EVMON_FORMAT_UE_TO_TABLES (
  'LOCKING', NULL, NULL, NULL, NULL, NULL, 'RECREATE_ONERROR', -1,
  'SELECT * FROM LOCK
     ORDER BY event_timestamp')
```

The results of the call are as follows:

1. The XSR object Paul.EVMON_LOCKING_SCHEMA_SQL1000 does not exist.
2. As the RECREATE_ONERROR option was specified, the tables are dropped and re-created.

Example 7: Using the previous relational tables on a release update

Greg has upgraded to a new DB2 release and does not want to pick up the new changes in the event monitor XML schema file. Greg calls the procedure with the *xsrobjectname* value from the previous release.

```
EVMON_FORMAT_UE_TO_TABLES (
  'LOCKING', NULL, 'EVMON_LOCKING_SCHEMA_SQL09070', NULL, NULL, NULL, NULL, -1,
  'SELECT * FROM LOCK
     ORDER BY event_timestamp')
```

Information returned

There is no output from the procedure except the SQLCA. The SQLCA indicates the completion status. The possible SQLCODES are:

0 All events were successfully inserted into the relational tables.

16278 One or more events were not inserted into the relational tables. The tokens within the SQLCA contain the total number of documents that were attempted and the total number of documents that succeeded decomposition.

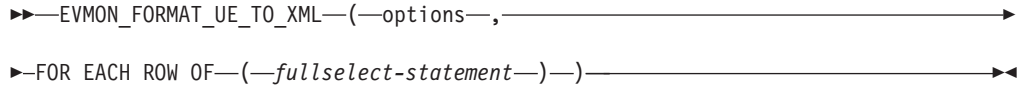
A diagnostic file is also created; and the name and location of that diagnostic file is stored in the db2diag log files, located in the DB2 diagnostic path.

negative sqlcode
An error has occurred, and investigating the SQLCODE message can provide additional details regarding the failure. For additional diagnostic messages, see the `db2diag` log files located in the DB2 diagnostic path.

EVMON_FORMAT_UE_TO_XML table function - convert unformatted events to XML

The EVMON_FORMAT_UE_TO_XML table function extracts binary events from an unformatted event table and formats them into an XML document.

Syntax



The schema is SYSPROC.

Table function parameters

options
An input argument of type VARCHAR(1024) that represents a list of keyword options supported by this table function.

LOG_TO_FILE
Indicates that the table function is to write the XML document to a file if the XML document is greater than 100 MB. The maximum size of each document returned by this table function per row is 100 MB. The file is written to the `<xml_document_id>.xml` file, where `<xml_document_id>` is the unique ID generated for each document. The output file is written to the DB2 diagnostic path directory.

LOG_PARTIAL_EVENTS
Indicates that the table function is to write all partial (incomplete) events to a file. A diagnostic message pointing to the specific file is inserted into the db2diag log files.

NULL No options selected.

fullselect-statement
The fullselect statement is a query that conforms to the rules of the SELECT statement. The query must follow the following rules:

- The query must use the "*" clause or specify all the columns of the unformatted event table. Otherwise an error is returned. The columns must be specified in the same order as returned by the DESCRIBE statement of the unformatted event table.
- The query must select only from an unformatted event table.
- The WHERE clause can use any of the non-LOB columns of the unformatted event table to filter out events.
- The SELECT statement must be specified by the keyword FOR EACH ROWS OF, enclosed in brackets.

Authorization

EXECUTE privilege on the EVMON_FORMAT_UE_TO_XML function.

SELECT privilege on the unformatted event table.

Examples

Example 1: Query all events from the unformatted event table "MYLOCKS".

```
SELECT evmon.* FROM TABLE (
  EVMON_FORMAT_UE_TO_XML (
    NULL,
    FOR EACH ROW OF (
      select * from MYLOCKS
        order by EVENT_TIMESTAMP )))
  AS evmon;
```

Example 2: Query all events of type "LOCKWAIT" that have occurred in the last 5 hours from the unformatted event table "LOCK".

```
SELECT evmon.* FROM TABLE (
  EVMON_FORMAT_UE_TO_XML (
    NULL,
    FOR EACH ROW OF (
      select * from LOCK order by EVENT_TIMESTAMP
        where EVENT_TYPE = 'LOCKWAIT'
        and EVENT_TIMESTAMP >= CURRENT_TIMESTAMP – 5 hours )))
  AS evmon;
```

Example 3: Get all events that belong to workload "PAYROLL" that occurred in the last 32 hours from the unformatted event table "UOW". Write the result to a file if any document is greater than 100 MB.

```
SELECT evmon.* FROM TABLE (
  EVMON_FORMAT_UE_TO_XML(
    'LOG TO FILE',
    FOR EACH ROW OF (
      select * from UOW order by EVENT_TIMESTAMP
        where WORKLOAD_NAME = 'PAYROLL'
        and EVENT_TIMESTAMP = CURRENT_TIMESTAMP - 32 hours )))
  AS evmon;
```

Example 4: Query all unit of work events from the "UOWEVMON" table, and use the XMLTABLE table function to present the UOW ID, UOW start and stop times, and the user ID for the person who issued the unit of work.

```
SELECT EVENT.UOW_ID, EVENT.APPLICATION_ID, EVENT.SESSION_AUTHID,
  EVENT.START_TIME, EVENT.STOP_TIME
  FROM TABLE(
    EVMON_FORMAT_UE_TO_XML(
      'LOG TO FILE',
      FOR EACH ROW OF(
        select * from UOWEVMON )))
    AS UEXML,
    XMLTABLE(
      XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon' ),
      '$uowevent/db2_uow_event'
      PASSING XMLPARSE( DOCUMENT UEXML.XMLREPORT ) as "uowevent"
      COLUMNS UOW_ID INTEGER PATH 'uow_id',
      MEMBER SMALLINT PATH '@member',
      APPLICATION_ID VARCHAR(128) PATH 'application_id',
      SESSION_AUTHID VARCHAR(128) PATH 'session_authid',
```

```
  START_TIME TIMESTAMP PATH 'start_time',
  STOP_TIME TIMESTAMP PATH 'stop_time'
)
AS EVENT
```

Usage notes

The EVMON_FORMAT_UE_TO_XML table function can be used with the lock and unit of work event monitors to extract data from an unformatted event table.

Depending on the event monitor type, the table function maps multiple records from the unformatted event table into a single event. In such a case, the records are cached in memory until all the records that make up the event are received. A larger memory requirement might result if the records passed into the table function are not in the order they were created and inserted into the table. If the records are not sorted in this manner, the table function must cache records for multiple events. To avoid this issue, qualify the *fullselect-statement* parameter with an ORDER BY clause that contains the following columns: EVENT_ID, EVENT_TIMESTAMP, EVENT_TYPE, and MEMBER. Memory consumption is reduced because at any particular time, the table function is processing and caching records from only a single event.

Information returned

Table 95. Information returned for EVMON_FORMAT_UE_TO_XML.

Column Name	Data Type	Description or corresponding monitor element
XMLID	VARCHAR(1024)	A unique document ID. The ID is derived as follows:<event_header>_<event_id>_<event_type>_<event_timestamp>_<partition>
XMLREPORT	BLOB(100M)	An XML document containing a single complete event. Each document has a maximum size of 100 MB.

MON_BP_UTILIZATION - Retrieve metrics for bufferpools

The MON_BP_UTILIZATION administrative view returns key monitoring metrics, including hit ratios and average read and write times, for all buffer pools and all database partitions in the currently connected database. It provides information that is critical for performance monitoring, because it helps you check how efficiently you are using your buffer pools.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7 , it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7 , this routine will no longer work.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_BP_UTILIZATION administrative view
- CONTROL privilege on the MON_BP_UTILIZATION administrative view

Information returned

Table 96. Information returned by the MON_BP_UTILIZATION administrative view

Column name	Data type	Description or Monitor element
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
MEMBER	SMALLINT	member - Database member
DATA_PHYSICAL_READS	BIGINT	Indicates the number of data pages read from the table space containers (physical) for temporary as well as regular and large table spaces. This is calculated as (*pool_data_p_reads* + *pool_temp_data_p_reads*) where *pool_data_p_reads* and *pool_temp_data_p_reads* represent the following monitor elements: • pool_data_p_reads - Buffer pool data physical reads • pool_temp_data_p_reads - Buffer pool temporary data physical reads
DATA_HIT_RATIO_ PERCENT	DECIMAL(5,2)	Data hit ratio, that is, the percentage of time that the database manager did not need to load a page from disk to service a data page request.
INDEX_PHYSICAL_READS	BIGINT	Indicates the number of index pages read from the table space containers (physical) for temporary as well as regular and large table spaces. This is calculated as (*pool_index_p_reads* + *pool_temp_index_p_reads*) where *pool_index_p_reads* + *pool_temp_index_p_reads* represent the following monitor elements: • pool_index_p_reads - Buffer pool index physical reads • pool_temp_index_p_reads - Buffer pool temporary index physical reads
INDEX_HIT_RATIO_ PERCENT	DECIMAL(5,2)	Index hit ratio, that is, the percentage of time that the database manager did not need to load a page from disk to service an index data page request.

Table 96. Information returned by the MON_BP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
XDA_PHYSICAL_READS	BIGINT	Indicates the number of data pages for XML storage objects (XDAs) read from the table space containers (physical) for temporary as well as regular and large table spaces. This is calculated as (*pool_xda_p_reads* + *pool_temp_xda_p_reads*) where *pool_xda_p_reads* and *pool_temp_xda_p_reads* represent the following monitor elements: • pool_xda_p_reads - Buffer pool XDA data physical reads • pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
XDA_HIT_RATIO_PERCENT	DECIMAL(5,2)	Auxiliary storage objects hit ratio, that is, the percentage of time that the database manager did not need to load a page from disk to service a data page request for XML storage objects (XDAs).

Table 96. Information returned by the MON_BP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
TOTAL_PHYSICAL_READS	BIGINT	Indicates the number of data pages, index pages, and data pages for XML storage objects (XDAs) read from the table space containers (physical) for temporary as well as regular and large table spaces. This is calculated as (*pool_data_p_reads* + *pool_temp_data_p_reads* + *pool_index_p_reads* + *pool_temp_index_p_reads* + *pool_xda_p_reads* + *pool_temp_xda_p_reads*) where *pool_data_p_reads*, *pool_temp_data_p_reads*, *pool_index_p_reads*, *pool_temp_index_p_reads*, *pool_xda_p_reads* and *pool_temp_xda_p_reads* represent the following monitor elements: • pool_data_p_reads - Buffer pool data physical reads • pool_temp_data_p_reads - Buffer pool temporary data physical reads • pool_index_p_reads - Buffer pool index physical reads • pool_temp_index_p_reads - Buffer pool temporary index physical reads • pool_xda_p_reads - Buffer pool XDA data physical reads • pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads

Table 96. Information returned by the MON_BP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
AVG_PHYSICAL_ READ_TIME	BIGINT	Average time, in milliseconds, spent reading pages from the table space containers (physical) for all types of table spaces. If the sum of physical reads is greater than zero, this is calculated as *pool_read_time* / (*pool_data_p_reads* + *pool_temp_data_p_reads* + *pool_index_p_reads* + *pool_temp_index_p_reads* + *pool_xda_p_reads* + *pool_temp_xda_p_reads*) where *pool_read_time*, *pool_data_p_reads*, *pool_temp_data_p_reads*, *pool_index_p_reads*, *pool_temp_index_p_reads*, *pool_xda_p_reads* and *pool_temp_xda_p_reads* represent the following monitor elements: • pool_read_time - Total buffer pool physical read time • pool_data_p_reads - Buffer pool data physical reads • pool_temp_data_p_reads - Buffer pool temporary data physical reads • pool_index_p_reads - Buffer pool index physical reads • pool_temp_index_p_reads - Buffer pool temporary index physical reads • pool_xda_p_reads - Buffer pool XDA data physical reads • pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads If the sum of physical reads is not greater than zero, NULL is returned.

Table 96. Information returned by the MON_BP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
PREFETCH_RATIO_ PERCENT	DECIMAL(5,2)	Percentage of pages read asynchronously (with prefetching). If many applications are reading data synchronously without prefetching, your system might not be tuned optimally.
ASYNC_NOT_READ_ PERCENT	DECIMAL(5,2)	Percentage of pages read asynchronously from disk, but never accessed by a query. If too many pages are read asynchronously from disk into the bufferpool, but no query ever accesses those pages, the prefetching might degrade performance. If the sum of asynchronous reads is greater than zero, this is calculated as *unread_prefetch_pages* / (*pool_async_data_reads* + *pool_async_index_reads* + *pool_async_xda_reads*) where *unread_prefetch_pages*, *pool_async_data_reads*, *pool_async_index_reads* and *pool_async_xda_reads* represent the following monitor elements: • unread_prefetch_pages - Unread prefetch pages • pool_async_data_reads - Buffer pool asynchronous data reads • pool_async_index_reads - Buffer pool asynchronous index reads • pool_async_xda_reads - Buffer pool asynchronous XDA data reads If the sum of asynchronous reads is not greater than zero, NULL is returned.

Table 96. Information returned by the MON_BP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
TOTAL_WRITES	BIGINT	The number of times a data, index, or data page for an XML storage object (XDA) was physically written to disk. This is calculated as (*pool_data_writes* + *pool_index_writes* + *pool_xda_writes*) where *pool_data_writes*, *pool_index_writes*, and *pool_xda_writes* represent the following monitor elements: • pool_data_writes - Buffer pool data writes • pool_index_writes - Buffer pool index writes • pool_xda_writes - Buffer pool XDA data writes
AVG_WRITE_TIME	BIGINT	Average time, in milliseconds, spent physically writing pages from the buffer pool to disk. If the sum of write operations is greater than zero, this is calculated as *pool_write_time* / (*pool_data_writes* + *pool_index_writes* + *pool_xda_writes*) where *pool_write_time*, *pool_data_writes*, *pool_index_writes*, and *pool_xda_writes* represent the following monitor elements: • pool_write_time - Total buffer pool physical write time • pool_data_writes - Buffer pool data writes • pool_index_writes - Buffer pool index writes • pool_xda_writes - Buffer pool XDA data writes If the sum of write operations is not greater than zero, NULL is returned.
SYNC_WRITES_PERCENT	DECIMAL(5,2)	Percentage of write operations that are synchronous.

MON_CONNECTION_SUMMARY - Retrieve metrics for all connections

The MON_CONNECTION_SUMMARY administrative view returns key metrics for all connections in the currently connected database. It is designed to help monitor the system in a high-level manner, showing incoming work per connection.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7 , it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

The metrics returned represent the accumulation of all metrics for requests that were submitted by the identified connection across all members of the database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_CONNECTION_SUMMARY administrative view
- CONTROL privilege on the MON_CONNECTION_SUMMARY administrative view
- DATAACCESS authority

Information returned

Table 97. Information returned by the MON_CONNECTION_SUMMARY administrative view

Column name	Data type	Description or Monitor element
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
APPLICATION_NAME	VARCHAR(128)	appl_name - Application name
APPLICATION_ID	VARCHAR(128)	appl_id - Application ID
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
TOTAL_APP_COMMITS	BIGINT	Total number of application commits across all members of the database for the specified service subclass.
TOTAL_APP_ROLLBACKS	BIGINT	Total number of application rollbacks across all members of the database for the specified service subclass.
ACT_COMPLETED_TOTAL	BIGINT	Total number of coordinator activities at any nesting level that completed successfully across all members of the database for the specified service subclass.

Table 97. Information returned by the MON_CONNECTION_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
APP_RQSTS_COMPLETED_TOTAL	BIGINT	Total number of external (application) requests that completed successfully across all members of the database for the specified service subclass
AVG_RQST_CPU_TIME	BIGINT	Average amount of CPU time, in microseconds, used by all external requests that completed successfully. It represents the total of both user and system CPU time.
ROUTINE_TIME_RQST_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent working on requests that was spent executing user routines.
RQST_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent working on requests that was spent waiting within the DB2 database server.
ACT_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent executing activities that was spent waiting within the DB2 database server.
IO_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent waiting within the DB2 database server that was due to I/O operations. This includes time spent performing direct reads or direct writes, and time spent reading data and index pages from the table space to the bufferpool or writing them back to disk.
LOCK_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent waiting on locks.
AGENT_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent by an application queued to wait for an agent under concentrator configurations.

Table 97. Information returned by the MON_CONNECTION_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
NETWORK_WAIT_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent on client-server communications. This includes time spent sending and receiving data over TCP/IP or using the IPC protocol.
SECTION_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent executing sections. This includes the time spent performing sorts.
SECTION_SORT_ PROC_TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing sorts while executing sections.
COMPILE_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent compiling an SQL statement. This includes explicit and implicit compile times.
TRANSACT_END_PROC _TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing commit processing or rolling back transactions.
UTILS_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent running utilities. This includes performing runstats, reorganization, and load operations.
AVG_LOCK_WAITS _PER_ACT	BIGINT	The average number of times that applications or connections waited for locks per coordinator activities (successful and aborted).
AVG_LOCK_TIMEOUTS _PER_ACT	BIGINT	The average number of times that a request to lock an object timed out per coordinator activities (successful and aborted).

Table 97. Information returned by the MON_CONNECTION_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
AVG_DEADLOCKS_ PER_ACT	BIGINT	The average number of deadlocks per coordinator activities (successful and aborted).
AVG_LOCK_ESCALS _PER_ACT	BIGINT	The average number of times that locks have been escalated from several row locks to a table lock per coordinator activities (successful and aborted).
ROWS_READ_PER_ ROWS_RETURNED	BIGINT	The average number of rows read from the table per rows returned to the application.
TOTAL_BP_HIT_ RATIO_PERCENT	DECIMAL(5,2)	The percentage of time that the database manager did not need to load a page from disk to service a data or index page request, including requests for XML storage objects (XDAs).

MON_CURRENT_SQL - Retrieve key metrics for all activities on all members

The MON_CURRENT_SQL administrative view returns key metrics for all activities that were submitted on all members of the database and have not yet been completed, including a point-in-time view of currently executing SQL statements (both static and dynamic) in the currently connected database.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7 , it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7 , this routine will no longer work.

You can use the MON_CURRENT_SQL administrative view to identify long running activities and prevent performance problems.

This view represents the coordinator perspective, and not that of individual members.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_CURRENT_SQL administrative view
- CONTROL privilege on the MON_CURRENT_SQL administrative view
- DATAACCESS authority

Information returned

Table 98. Information returned by the MON_CURRENT_SQL administrative view

Column name	Data type	Description or Monitor element
COORD_MEMBER	SMALLINT	coord_member - Coordinating member
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
APPLICATION_NAME	VARCHAR(128)	appl_name - Application name
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
CLIENT_APPLNAME	VARCHAR(128)	CURRENT CLIENT_APPLNAME special register
ELAPSED_TIME_SEC	INTEGER	The time elapsed since this activity began, in seconds. The value of this column is null when an activity has entered the system but is in a queue and has not started running.
ACTIVITY_STATE	VARCHAR(32)	activity_state - Activity state
ACTIVITY_TYPE	VARCHAR(32)	activity_type - Activity type
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_RETURNED	BIGINT	rows_returned - Rows returned
QUERY_COST_ESTIMATE	BIGINT	query_cost_estimate - Query cost estimate
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
STMT_TEXT	CLOB(2MB)	stmt_text - SQL statement text

MON_CURRENT_UOW - Retrieve metrics for all units of work

The MON_CURRENT_UOW administrative view returns key metrics for all units of work that were submitted on all members of the database. It identifies long running units of work and can therefore be used to prevent performance problems.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7 , it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7 , this routine will no longer work.

The MON_CURRENT_UOW view represents the coordinator perspective, and not individual members.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_CURRENT_UOW administrative view
- CONTROL privilege on the MON_CURRENT_UOW administrative view
- DATAACCESS authority

Example

The following example retrieves the application handle, the unit of work ID, the elapsed time, and the total number of rows read and rows returned, for all units of work that have been executed for more than 1 minute.

```
SELECT APPLICATION_HANDLE AS APPL_HANDLE,
       UOW_ID, ELAPSED_TIME_SEC,
       TOTAL_ROWS_MODIFIED AS TOTAL_READ,
       TOTAL_ROWS_MODIFIED AS TOTAL_MODIFIED
  FROM MON_CURRENT_UOW
 WHERE ELAPSED_TIME_SEC > 60
 ORDER BY ELAPSED_TIME_SEC DESC
```

The following is an example of output for this query.

```
APPL_HANDLE UOW_ID ELAPSED_TIME_SEC TOTAL_READ TOTAL_MODIFIED
----------- ------ ---------------- ---------- --------------
        254      1              750      87460              0
         61      1              194        108              0
        145      4               82          0             34

  3 record(s) selected.
```

Information returned

Table 99. Information returned by the MON_CURRENT_UOW administrative view

Column name	Data type	Description or Monitor element
COORD_MEMBER	SMALLINT	coord_member - Coordinator member
UOW_ID	INTEGER	uow_id - Unit of work ID
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
APPLICATION_NAME	VARCHAR(128)	appl_name - Application name
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
CLIENT_APPLNAME	VARCHAR(255)	CURRENT CLIENT_APPLNAME special register

Table 99. Information returned by the MON_CURRENT_UOW administrative view (continued)

Column name	Data type	Description or Monitor element
ELAPSED_TIME_SEC	INTEGER	The time elapsed since this unit of work began, in seconds. The value of the column is NULL when an activity has entered the system but is in a queue and has not started running.
WORKLOAD_OCCURRENCE_STATE	VARCHAR(32)	workload_occurrence_state - Workload occurrence state
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
TOTAL_ROWS_MODIFIED	BIGINT	The total number of rows inserted, updated or deleted.
TOTAL_ROWS_READ	BIGINT	The total number of rows read from tables.
TOTAL_ROWS_RETURNED	BIGINT	The total number of rows that have been selected and returned to the application.

MON_DB_SUMMARY - Retrieve accumulated metrics across all members of the database

The MON_DB_SUMMARY administrative view returns key metrics aggregated over all service classes in the currently connected database. It is designed to help monitor the system in a high-level manner by providing a concise summary of the database.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7 , it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7 , this routine will no longer work.

The metrics returned represent the accumulation of metrics across all members of the database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_DB_SUMMARY administrative view
- CONTROL privilege on the MON_DB_SUMMARY administrative view
- DATAACCESS authority

Information returned

Table 100. Information returned by the MON_DB_SUMMARY administrative view

Column name	Data type	Description or Monitor element
TOTAL_APP_COMMITS	BIGINT	Total number of application commits across all members of the database for the specified service subclass.
TOTAL_APP_ROLLBACKS	BIGINT	Total number of application rollbacks across all members of the database for the specified service subclass.
ACT_COMPLETED_TOTAL	BIGINT	Total number of coordinator activities at any nesting level that completed successfully across all members of the database for the specified service subclass.
APP_RQSTS_COMPLETED _TOTAL	BIGINT	Total number of external (application) requests that completed successfully across all members of the database for the specified service subclass
AVG_RQST_CPU_TIME	BIGINT	Average amount of CPU time, in microseconds, used by all external requests that completed successfully. It represents the total of both user and system CPU time.
ROUTINE_TIME_ RQST_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent working on requests that was spent executing user routines.
RQST_WAIT_ TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent working on requests that was spent waiting within the DB2 database server.
ACT_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of the time spent executing activities that was spent waiting within the DB2 database server.
IO_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent waiting within the DB2 database server that was due to I/O operations. This includes time spent performing direct reads or direct writes, and time spent reading data and index pages from the table space to the bufferpool or writing them back to disk.

Table 100. Information returned by the MON_DB_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
LOCK_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent waiting on locks.
AGENT_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent by an application queued to wait for an agent under concentrator configurations.
NETWORK_WAIT_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent on client-server communications. This includes time spent sending and receiving data over TCP/IP or using the IPC protocol.
SECTION_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent executing sections. This includes the time spent performing sorts.
SECTION_SORT_ PROC_TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing sorts while executing sections.
COMPILE_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent compiling an SQL statement. This includes explicit and implicit compile times.
TRANSACT_END_PROC _TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing commit processing or rolling back transactions.

Table 100. Information returned by the MON_DB_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
UTILS_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent running utilities. This includes performing runstats, reorganization, and load operations.
AVG_LOCK_WAITS _PER_ACT	BIGINT	The average number of times that applications or connections waited for locks per coordinator activities (successful and aborted).
AVG_LOCK_TIMEOUTS _PER_ACT	BIGINT	The average number of times that a request to lock an object timed out per coordinator activities (successful and aborted).
AVG_DEADLOCKS_ PER_ACT	BIGINT	The average number of deadlocks per coordinator activities (successful and aborted).
AVG_LOCK_ESCALS _PER_ACT	BIGINT	The average number of times that locks have been escalated from several row locks to a table lock per coordinator activities (successful and aborted).
ROWS_READ_PER_ ROWS_RETURNED	BIGINT	The average number of rows read from the table per rows returned to the application.
TOTAL_BP_HIT_ RATIO_PERCENT	DECIMAL(5,2)	The percentage of time that the database manager did not need to load a page from disk to service a data or index page request, including requests for XML storage objects (XDAs).

MON_FORMAT_LOCK_NAME - format the internal lock name and return details

The MON_FORMAT_LOCK_NAME table function formats the internal lock name and returns details regarding the lock in a row-based format. Each returned row consists of a *key-value* pair relevant for that particular lock.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command

(because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

To get information about locks, use the MON_FORMAT_LOCK_NAME, MON_GET_LOCKS, and, MON_GET_APPL_LOCKWAIT table functions instead of the SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function, and the SNAPLOCK administrative view and SNAP_GET_LOCK table function, which are deprecated in Fixpack 1 of Version 9.7.

The schema is SYSPROC.

Table function parameters

lockname
An input argument of type VARCHAR(32) that specifies the internal binary name of the lock that is to be formatted. A NULL value results in error SQL0171N being returned.

Authorization

The following privilege is required:

- EXECUTE privilege on the MON_FORMAT_LOCK_NAME table function

Examples

The internal lock name is returned in a variety of situations, such as being written to the `db2diag log` files, or as the value of the **lock_name** monitor element. The following example shows how to use the MON_FORMAT_LOCK_NAME table function to find out further information about the lock, in this case with a lock name of 0000000E00000000000B00C152.

```
SELECT SUBSTR(NAME,1,20) AS NAME,
       SUBSTR(VALUE,1,50) AS VALUE
       FROM
       TABLE( MON_FORMAT_LOCK_NAME('0000000E00000000000B00C152')) as LOCK
```

The following output is returned:

```
NAME                 VALUE
---------------      ------------------------------------------------------
LOCK_OBJECT_TYPE     ROW
TBSP_NAME            SYSCATSPACE
TABSCHEMA            SYSIBM
TABNAME              SYSPLANDEP
ROWID                0000000B00C1

 4 record(s) selected
```

Information returned

Table 101. Information returned by the MON_FORMAT_LOCK_NAME table function

Column name	Data type	Description
NAME	VARCHAR(256)	Element of the lock name. See table below for more details.
VALUE	VARCHAR(1024)	Value of the element.

Not all elements that make up the specified lock name are returned; only those *key-value* pairs that are relevant are returned.

The elements that can be returned are as follows:

Table 102. Monitor elements that can be returned

Element name	Description	Possible values or monitor element
LOCK_OBJECT_TYPE	The lock object type	lock_object_type - Lock object type For possible values, see "lock_object_type - Lock object type waited on monitor element".
DATA_PARTITION_ID	The identifier of the data partition for which information is returned. This element is only applicable to partitioned tables and partitioned indexes. When returning lock level information, a value of -1 represents a lock which controls access to the whole table.	data_partition_id - Data partition identifier
TBSP_NAME	The name of a table space	tablespace_name - Table space name
TABSCHEMA	The schema of the table	table_schema - Table schema name
TABNAME	The name of the table	table_name - Table name
ROWID	Row ID of the table	-
PAGEID	The page ID	-
WORKLOAD_NAME	Name of the workload	workload_name - Workload name
STORAGE_GRP_ID	The storage group ID	-
BUFFERPOOL_NAME	Name of the buffer pool	-
FED_SERVER_NAME	Name of the federation server	-
FED_USER_NAME	Name of the federation user mapping	-

Table 102. Monitor elements that can be returned (continued)

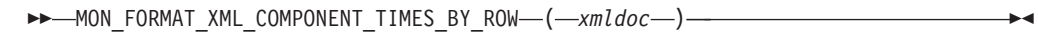
Element name	Description	Possible values or monitor element
SEQ_OPERATION	Operation requesting a sequence lock	Possible values are: • AUTONOMIC_POLICIES • CATALOG_ARRAY • DESCRIBE • INIT_EVMON • INIT_PACKAGE • INIT_AUDIT • PACKAGE_CREATION • INIT_ROUTINE_ID • INIT_ROLE_ID • TEMP_TBSPACE • AUDIT_DDL • VERSION_TIMES • WLM • TRUSTED_CTX • INIT_TRUSTED_CTX • STATIC_STMT • USER_TEMP_TBSPACE
CONTAINER_ID	The container ID	-
STMT_UID	The statement ID	-
PACKAGE_TOKEN	The package token	-
INTERNAL	Reserved for internal use	-

MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW - Get formatted row-based component times

The MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW table function returns formatted row-based output for the component times contained in an XML metrics document.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

Syntax



The schema is SYSPROC.

Table function parameters

xmldoc
An input argument of type BLOB(100M) that contains an XML document with

either a system_metrics or activity_metrics element. XML documents with these elements can be obtained from the following sources:

- Returned by one of the MON_GET_*_DETAILS table functions.
- From the metrics column output by statistics and activity event monitors.
- From the formatted output of the unit of work, or package cache event monitors.

Authorization

EXECUTE privilege on the
MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW function.

Example

The following example returns the breakdown of component times within the DB2 database manager for service subclasses, which shows both the total time spent in any given component, as well as the amount of time that was actually spent processing, rather than waiting, in a component.

```
SELECT SUBSTR(T.SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS,
  SUBSTR(T.SERVICE_SUBCLASS_NAME,1,19) AS SUBCLASS,
  T.MEMBER,
  SUBSTR(COMP.METRIC_NAME,1,15) AS METRIC_NAME
  SUBSTR(COMP.PARENT_METRIC_NAME,1,15) AS PARENT_NAME
  COMP.TOTAL_TIME_VALUE AS TOTAL_TIME,
  COMP.PROC_TIME_VALUE AS TOTAL_PROC_TIME,
   COMP.COUNT
 FROM TABLE (MON_GET_SERVICE_SUBCLASS_DETAILS(NULL,
              NULL,-2)) AS T,
       TABLE(MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW(
                                          T.DETAILS
                                         )) AS COMP

 WHERE COMP.PARENT_METRIC_NAME IS NOT NULL;
```

The following is an example of output from this query.

```
SUPERCLASS          SUBCLASS            MEMBER METRIC_NAME     PARENT_NAME    ...
------------------- ------------------- ------ --------------- ---------------...
MYSC                MYSSC                    0 TOTAL_COMPILE_T TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_IMPLICIT_ TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_SECTION_T TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_COMMIT_TI TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_ROLLBACK_ TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_RUNSTATS_ TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_REORG_TIM TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_LOAD_TIME TOTAL_RQST_TIME...
MYSC                MYSSC                    0 TOTAL_SECTION_S TOTAL_SECTION_T...

  9 record(s) selected.
```

The following is a continuation of sample output from this query.

```
...TOTAL_TIME           TOTAL_PROC_TIME      COUNT
...--------------------- -------------------- --------------------
...                 100                  100                    1
...                   0                    0                    0
...                1253                  953                    0
...                 213                  153                    0
...                   0                    0                    0
...                   0                    0                    0
...                   0                    0                    0
```

```
...                  0                    0                    0
...                  0                    0                    0

  9 record(s) selected.
```

Information returned

Table 103. Information returned for MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW

Column Name	Data Type	Description
METRIC_NAME	VARCHAR(128)	The unique identifier for the total time metric value.
PROC_METRIC_NAME	VARCHAR(128)	The unique identifier for the processing time metric.
TOTAL_TIME_VALUE	BIGINT	The total time value in milliseconds corresponding to metric_name.
PROC_TIME_VALUE	BIGINT	The processing time value in milliseconds corresponding to proc_metric_name
COUNT	BIGINT	Number of occurrences of this type of interval.
PARENT_METRIC_NAME	VARCHAR(128)	The identifier of the parent total time metric whose value contains the total_time_value as a subset
PARENT_PROC_ METRIC_NAME	VARCHAR(128)	The identifier of the parent processing time metric whose value contains the proc_time_value as a subset

XML documents that contain an element of type *system_metrics* are generated from the following interfaces:

- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_UNIT_OF_WORK_DETAILS
- MON_GET_WORKLOAD_DETAILS
- DETAILS_XML column from a STATISTICS event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the UNIT OF WORK event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the UNIT OF WORK event monitor

See Table 104 for the types of metrics and their parent metrics that are returned from the XML document in this case:

Table 104. Metric names returned by MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW for XML documents containing a system_metrics element type

Metric Name	Proc Metric Name	Parent Metric Name	Parent Proc Metric Name	Description of metric or Monitor element
TOTAL_ RQST_TIME	NULL	NULL	NULL	total_rqst_time - Total request time
TOTAL_COMPILE _TIME	TOTAL_ COMPILE_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_compile_time - Total compile time
TOTAL_IMPLICIT_ COMPILE_TIME	TOTAL_ IMPLICIT_ COMPILE_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_implicit_compile_time - Total implicit compile time

Table 104. Metric names returned by MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW for XML documents containing a system_metrics element type (continued)

Metric Name	Proc Metric Name	Parent Metric Name	Parent Proc Metric Name	Description of metric or Monitor element
TOTAL_SECTION_ TIME	TOTAL_ SECTION_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_section_time - Total section time
TOTAL_COMMIT_ TIME	TOTAL_ COMMIT_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_commit_time - Total commit time
TOTAL_ROLLBACK _TIME	TOTAL_ ROLLBACK_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_rollback_time - Total rollback time
TOTAL_ROUTINE_ USER_CODE_ TIME	TOTAL_ ROUTINE_USER_ CODE_PROC_ TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_routine_user_code_time - Total routine user code time
TOTAL_RUNSTATS_ TIME	TOTAL_ RUNSTATS_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_runstats_time - Total runtime statistics
TOTAL_REORG_ TIME	TOTAL_ REORG_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_reorg_time - Total reorganization time
TOTAL_LOAD_ TIME	TOTAL_ LOAD_ PROC_TIME	TOTAL_ RQST_TIME	TOTAL_ RQST_TIME	total_load_time - Total load time
TOTAL_SECTION_ SORT_TIME	TOTAL_ SECTION_ SORT_ PROC_TIME	TOTAL_ SECTION_TIME	TOTAL_ SECTION_ PROC_TIME	total_section_sort_time - Total section sort time

XML documents that contain an element of type *activity_metrics* are generated from the following interfaces:

- MON_GET_ACTIVITY_DETAILS
- MON_GET_PKG_CACHE_STMT_DETAILS
- DETAILS_XML column from an ACTIVITY event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the PACKAGE CACHE event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the PACKAGE CACHE event monitor

See Table 105 for the types of metrics and their parent metrics that are returned from the XML document in this case:

Table 105. Metric names returned by MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW for XML documents containing an activity_metrics element type

Metric Name	Proc Metric Name	Parent Metric Name	Parent Proc Metric Name	Description or Monitor element
STMT_ EXEC_TIME	NULL	NULL	NULL	stmt_exec_time - Statement execution time

Table 105. Metric names returned by MON_FORMAT_XML_COMPONENT_TIMES_BY_ROW for XML documents containing an activity_metrics element type (continued)

Metric Name	Proc Metric Name	Parent Metric Name	Parent Proc Metric Name	Description or Monitor element
TOTAL_ ROUTINE_ TIME	NULL	STMT_ EXEC_TIME	NULL	total_routine_time - Total routine time
TOTAL_ ROUTINE_ NON_SECT_ TIME	TOTAL_ ROUTINE_ NON_SECT_ PROC_ TIME	TOTAL_ ROUTINE_ TIME	STMT_ EXEC_TIME	total_routine_non_sect_time - Non-section routine execution time
TOTAL_ ROUTINE_USER_ CODE_TIME	TOTAL_ ROUTINE_ USER_ CODE_ PROC_TIME	TOTAL_ ROUTINE_ NON_SECT_ TIME	TOTAL_ ROUTINE_ NON_ SECT_PROC_ TIME	total_routine_user_code_time - Total routine user code time
TOTAL_ SECTION_ TIME	TOTAL_ SECTION_ PROC_TIME	STMT_ EXEC_TIME	STMT_ EXEC_TIME	total_section_time - Total section time
TOTAL_ SECTION_ SORT_TIME	TOTAL_ SECTION_ SORT_PROC_ TIME	TOTAL_ SECTION_ TIME	TOTAL_ SECTION_ PROC_TIME	total_section_sort_time - Total section sort time

MON_FORMAT_XML_METRICS_BY_ROW - Get formatted row-based output for all metrics

The MON_FORMAT_XML_METRICS_BY_ROW table function returns formatted row-based output for all metrics contained in an XML metrics document.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

Syntax

The schema is SYSPROC.

Table function parameters

xmldoc
An input argument of type BLOB(100M) that contains an XML document with either a system_metrics or activity_metrics element. XML documents with these elements can be obtained from the following sources:

- Returned by one of the MON_GET_*_DETAILS table functions.
- From the metrics column output by statistics and activity event monitors.

- From the formatted output of the unit of work, or package cache event monitors.

Authorization

EXECUTE privilege on the MON_FORMAT_XML_METRICS_BY_ROW function.

Example

This example shows how to call the MON_FORMAT_XML_METRICS _BY_ROW table function to return row-based formatted information from the XML document produced by the MON_GET_WORKLOAD_DETAILS table function.

```
SELECT SUBSTR(TFXML.WORKLOAD_NAME, 1, 13) AS WORKLOAD_NAME,
       SUBSTR(METRICS.METRIC_NAME, 1, 25) AS METRIC_NAME,
       METRICS.VALUE
   FROM
     TABLE( MON_GET_WORKLOAD_DETAILS( NULL, -2 ) ) AS TFXML,
     TABLE( MON_FORMAT_XML_METRICS_BY_ROW(
                                           TFXML.DETAILS
                                         )) AS METRICS
ORDER BY METRICS.VALUE DESC
```

The following is a partial listing of the output of this query.

```
WORKLOAD_NAME METRIC_NAME               VALUE
------------- ------------------------- ---------------
PAYROLL       ACT_COMPLETED_TOTAL                    15
FINANCE       ACT_COMPLETED_TOTAL                    12
PAYROLL       LOCK_WAITS                              8
FINANCE       LOCK_WAITS                              5
FINANCE       DEADLOCKS                               3
PAYROLL       DEADLOCKS                               0
```

Information returned

Table 106. Information returned for MON_FORMAT_XML_METRICS_BY_ROW

Column Name	Data Type	Description
METRIC_NAME	VARCHAR(128)	The unique identifier for the total time metric value.
VALUE	BIGINT	The current value of the metric.

XML documents that contain an element of type *system_metrics* are generated from the following interfaces:

- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_UNIT_OF_WORK_DETAILS
- MON_GET_WORKLOAD_DETAILS
- DETAILS_XML column from a STATISTICS event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the UNIT OF WORK event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the UNIT OF WORK event monitor

See Table 107 on page 390 for the types of metrics that are returned from the XML document in this case:

Table 107. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing a system_metrics element type

Metric Name	Description of metric or Monitor element
TOTAL_WAIT_TIME	total_wait_time - Total wait time
CLIENT_IDLE_WAIT_TIME	client_idle_wait_time - Client idle wait time
POOL_READ_TIME	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	pool_write_time - Total buffer pool physical write time
DIRECT_READ_TIME	direct_read_time - Direct read time
DIRECT_WRITE_TIME	direct_write_time - Direct write time
LOCK_WAIT_TIME	lock_wait_time - Time waited on locks
AGENT_WAIT_TIME	agent_wait_time - Agent wait time
WLM_QUEUE_TIME_TOTAL	wlm_queue_time_total - Workload manager total queue time
FCM_SEND_WAIT_TIME	fcm_send_wait_time - FCM send wait time
FCM_RECV_WAIT_TIME	fcm_recv_wait_time - FCM recv wait time
TCPIP_SEND_WAIT_TIME	tcpip_send_wait_time - TCP/IP send wait time
TCPIP_RECV_WAIT_TIME	tcpip_recv_wait_time - TCP/IP recv wait time
IPC_SEND_WAIT_TIME	ipc_send_wait_time - Interprocess communication send wait time
IPC_RECV_WAIT_TIME	ipc_recv_wait_time - Interprocess communication recv wait time
LOG_BUFFER_WAIT_TIME	log_buffer_wait_time - Log buffer wait time
LOG_DISK_WAIT_TIME	log_disk_wait_time - Log disk wait time
FCM_MESSAGE_SEND_WAIT_TIME	fcm_message_send_wait_time - FCM message send wait time
FCM_MESSAGE_RECV_WAIT_TIME	fcm_message_recv_wait_time - FCM message recv wait time
FCM_TQ_SEND_WAIT_TIME	fcm_tq_send_wait_time - FCM tablequeue send wait time
FCM_TQ_RECV_WAIT_TIME	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
AUDIT_FILE_WRITE_WAIT_TIME	audit_file_write_wait_time - Audit file write wait time
AUDIT_SUBSYSTEM_WAIT_TIME	audit_subsystem_wait_time - Audit subsystem wait time
DIAGLOG_WRITE_WAIT_TIME	diaglog_write_wait_time - Diag log write time
TOTAL_RQST_TIME	total_rqst_time - Total request time
TOTAL_COMPILE_TIME	total_compile_time - Total compile time
TOTAL_IMPLICIT_COMPILE_TIME	total_implicit_compile_time - Total implicit compile time
TOTAL_SECTION_TIME	total_section_time - Total section time
TOTAL_COMMIT_TIME	total_commit_time - Total commit time
TOTAL_ROLLBACK_TIME	total_rollback_time - Total rollback time
TOTAL_RUNSTATS_TIME	total_runstats_time - Total runtime statistics
TOTAL_REORG_TIME	total_reorg_time - Total reorganization time
TOTAL_LOAD_TIME	total_load_time - Total load time
TOTAL_SECTION_SORT_TIME	total_section_sort_time - Total section sort time
TOTAL_ROUTINE_USER_CODE_TIME	total_routine_user_code_time - Total routine user code time
TOTAL_COMPILE_PROC_TIME	total_compile_proc_time - Total compile processing time
TOTAL_IMPLICIT_ COMPILE_PROC_TIME	total_implicit_compile_proc_time - Total implicit compile processing time
TOTAL_SECTION_PROC_TIME	total_section_proc_time - Total section processing time

Table 107. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing a system_metrics element type (continued)

Metric Name	Description of metric or Monitor element
TOTAL_COMMIT_PROC_TIME	total_commit_proc_time - Total commits processing time
TOTAL_ROLLBACK_PROC_TIME	total_rollback_proc_time - Total rollback processing time
TOTAL_RUNSTATS_PROC_TIME	total_runstats_proc_time - Total runtime statistics processing time
TOTAL_REORG_PROC_TIME	total_reorg_proc_time - Total reorganization processing time
TOTAL_LOAD_PROC_TIME	total_load_proc_time - Total load processing time
TOTAL_SECTION_SORT_PROC_TIME	total_section_sort_proc_time - Total section sort processing time
TOTAL_ROUTINE_USER_CODE_ PROC_TIME	total_routine_user_code_proc_time - Total routine user code processing time
ACT_ABORTED_TOTAL	act_aborted_total - Total aborted activities
ACT_COMPLETED_TOTAL	act_completed_total - Total completed activities
ACT_REJECTED_TOTAL	act_rejected_total - Total rejected activities
AGENT_WAITS_TOTAL	agent_waits_total - Total agent waits
POOL_DATA_L_READS	pool_data_l_reads - Buffer pool data logical reads
POOL_INDEX_L_READS	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_DATA_L_READS	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_INDEX_L_READS	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_XDA_L_READS	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_XDA_L_READS	pool_xda_l_reads - Buffer pool XDA data logical reads
POOL_DATA_P_READS	pool_data_p_reads - Buffer pool data physical reads
POOL_INDEX_P_READS	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_DATA_P_READS	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_P_READS	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_P_READS	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_XDA_P_READS	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_DATA_WRITES	pool_data_writes - Buffer pool data writes
POOL_INDEX_WRITES	pool_index_writes - Buffer pool index writes
POOL_XDA_WRITES	pool_xda_writes - Buffer pool XDA data writes
DEADLOCKS	deadlocks - Deadlocks detected
DIRECT_READS	direct_reads - Direct reads from database
DIRECT_WRITES	direct_writes - Direct writes to database
DIRECT_READ_REQS	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	direct_write_reqs - Direct write requests
FCM_RECV_VOLUME	fcm_recv_volume - FCM recv volume
FCM_RECVS_TOTAL	fcm_recvs_total - FCM recvs total
FCM_SEND_VOLUME	fcm_send_volume - FCM send volume
FCM_SENDS_TOTAL	fcm_sends_total - FCM sends total
IPC_RECV_VOLUME	ipc_recv_volume - Interprocess communication recv volume
IPC_RECVS_TOTAL	ipc_recvs_total - Interprocess communication recvs total
IPC_SEND_VOLUME	ipc_send_volume - Interprocess communication send volume

Table 107. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing a system_metrics element type (continued)

Metric Name	Description of metric or Monitor element
IPC_SENDS_TOTAL	ipc_sends_total - Interprocess communication send total
LOCK_ESCALS	lock_escals - Number of lock escalations
LOCK_TIMEOUTS	lock_timeouts - Number of lock timeouts
LOCK_WAITS	lock_waits - Lock waits
NUM_LOG_BUFFER_FULL	num_log_buffer_full - Number of full log buffers
LOG_DISK_WAITS_TOTAL	log_disk_waits_total - Log disk waits total
RQSTS_COMPLETED_TOTAL	rqsts_completed_total - Total requests completed
ROWS_MODIFIED	rows_modified - Rows modified
ROWS_READ	rows_read - Rows read
ROWS_RETURNED	rows_returned - Rows returned
TCPIP_RECV_VOLUME	tcpip_recv_volume - TCP/IP received volume
TCPIP_SEND_VOLUME	tcpip_send_volume - TCP/IP send volume
TCPIP_RECVS_TOTAL	tcpip_recvs_total - TCP/IP recvs total
TCPIP_SENDS_TOTAL	tcpip_sends_total - TCP/IP sends total
WLM_QUEUE_ASSIGNMENTS_TOTAL	wlm_queue_assignments_total - Workload manager total queue assignments
APP_RQSTS_COMPLETED_TOTAL	app_rqsts_completed_total - Total application requests completed
TOTAL_SECTION_SORTS	total_section_sorts - Total section sorts
TOTAL_SORTS	total_sorts - Total Sorts
POST_THRESHOLD_SORTS	post_threshold_sorts - Post threshold sorts
POST_SHRTHRESHOLD_SORTS	post_shrthreshold_sorts - Post shared threshold sorts
SORT_OVERFLOWS	sort_overflows - Sort overflows
ACT_RQSTS_TOTAL	act_rqsts_total - Total activity requests
TOTAL_ROUTINE_INVOCATIONS	total_routine_invocations - Total routine invocations
TOTAL_COMPILATIONS	total_compilations - Total compilations
TOTAL_IMPLICIT_COMPILATIONS	total_implicit_compilations - Total implicit complications
TOTAL_APP_SECTION_EXECUTIONS	total_app_section_executions - Total section executions
TOTAL_APP_COMMITS	total_app_commits - Total application commits
INT_COMMITS	int_commits - Internal commits
TOTAL_APP_ROLLBACKS	total_app_rollbacks - Total application rollbacks
INT_ROLLBACKS	int_rollbacks - Internal rollbacks
TOTAL_RUNSTATS	total_runstats - Total runtime statistics
TOTAL_REORGS	total_reorgs - Total reorganizations
TOTAL_LOADS	total_loads - Total loads
CAT_CACHE_INSERTS	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_LOOKUPS	cat_cache_lookups - Catalog cache lookups
PKG_CACHE_INSERTS	pkg_cache_inserts - Package cache inserts
PKG_CACHE_LOOKUPS	pkg_cache_lookups - Package cache lookups
THRESH_VIOLATIONS	thresh_violations - Number of threshold violations

Table 107. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing a system_metrics element type (continued)

Metric Name	Description of metric or Monitor element
NUM_LW_THRESH_EXCEEDED	num_lw_thresh_exceeded - Number of thresholds exceeded
AUDIT_EVENTS_TOTAL	audit_events_total - Total audit events
AUDIT_SUBSYSTEM_WAITS_TOTAL	audit_subsystem_waits_total - Total audit subsystem waits
AUDIT_FILE_WRITES_TOTAL	audit_file_writes_total - Total Audit files written
DIAGLOG_WRITES_TOTAL	diaglog_writes_total - Diag log total writes
FCM_MESSAGE_RECV_VOLUME	fcm_message_recv_volume - FCM message recv volume
FCM_MESSAGE_RECVS_TOTAL	fcm_message_recvs_total - FCM message recvs total
FCM_MESSAGE_SEND_VOLUME	fcm_message_send_volume - FCM message send volume
FCM_MESSAGE_SENDS_TOTAL	fcm_message_sends_total - FCM message sends total
FCM_TQ_RECV_VOLUME	fcm_tq_recv_volume - FCM tablequeue recv volume
FCM_TQ_RECVS_TOTAL	fcm_tq_recvs_total - FCM tablequeue recvs total
FCM_TQ_SEND_VOLUME	fcm_tq_send_volume - FCM tablequeue send volume
FCM_TQ_SENDS_TOTAL	fcm_tq_sends_total - FCM tablequeue send total
TQ_TOT_SEND_SPILLS	tq_tot_send_spills - Total number of table queue buffers overflowed
TOTAL_ROUTINE_TIME	total_routine_time - Total routine time
TOTAL_CPU_TIME	total_cpu_time - Total CPU time
TOTAL_ACT_TIME	total_act_time - Total activity time
TOTAL_ACT_WAIT_TIME	total_act_wait_time - Total activity wait time
TOTAL_APP_RQST_TIME	total_app_rqst_time - Total application request time

XML documents that contain an element of type *activity_metrics* are generated from the following interfaces:

- MON_GET_ACTIVITY_DETAILS
- MON_GET_PKG_CACHE_STMT_DETAILS
- DETAILS_XML column from an ACTIVITY event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the PACKAGE CACHE event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the PACKAGE CACHE event monitor

See Table 108 for the types of metrics that are returned from the XML document in this case:

Table 108. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing an activity_metrics element type

Metric Name	Description or Monitor element
TOTAL_ACT_WAIT_TIME	total_act_wait_time - Total activity wait time
POOL_READ_TIME	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	pool_write_time - Total buffer pool physical write time
DIRECT_READ_TIME	direct_read_time - Direct read time
DIRECT_WRITE_TIME	direct_write_time - Direct write time
WLM_QUEUE_TIME_TOTAL	wlm_queue_time_total - Workload manager total queue time

Table 108. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing an activity_metrics element type (continued)

Metric Name	Description or Monitor element
LOCK_WAIT_TIME	lock_wait_time - Time waited on locks
LOG_BUFFER_WAIT_TIME	log_buffer_wait_time - Log buffer wait time
LOG_DISK_WAIT_TIME	log_disk_wait_time - Log disk wait time
AUDIT_FILE_WRITE_WAIT_TIME	audit_file_write_wait_time - Audit file write wait time
AUDIT_SUBSYSTEM_WAIT_TIME	audit_subsystem_wait_time - Audit subsystem wait time
DIAGLOG_WRITE_WAIT_TIME	diaglog_write_wait_time - Diag log write time
FCM_SEND_WAIT_TIME	fcm_send_wait_time - FCM send wait time
FCM_RECV_WAIT_TIME	fcm_recv_wait_time - FCM recv wait time
FCM_MESSAGE_SEND_WAIT_TIME	fcm_message_send_wait_time - FCM message send wait time
FCM_MESSAGE_RECV_WAIT_TIME	fcm_message_recv_wait_time - FCM message recv wait time
FCM_TQ_SEND_WAIT_TIME	fcm_tq_send_wait_time - FCM tablequeue send wait time
FCM_TQ_RECV_WAIT_TIME	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
STMT_EXEC_TIME	stmt_exec_time - Statement execution time
TOTAL_ROUTINE_TIME	total_routine_time - Total routine time
TOTAL_ROUTINE_NON_SECT_TIME	total_routine_non_sect_time - Non-section routine execution time
TOTAL_ROUTINE_USER_CODE_TIME	total_routine_user_code_time - Total routine user code time
TOTAL_SECTION_TIME	total_section_time - Total section time
TOTAL_SECTION_SORT_TIME	total_section_sort_time - Total section sort time
TOTAL_ROUTINE_NON_ SECT_PROC_TIME	total_routine_non_sect_proc_time - Non-section processing time
TOTAL_ROUTINE_USER_CODE_ PROC_TIME	total_routine_user_code_proc_time - Total routine user code processing time
TOTAL_SECTION_PROC_TIME	total_section_proc_time - Total section processing time
TOTAL_SECTION_SORT_PROC_TIME	total_section_sort_proc_time - Total section sort processing time
TOTAL_SECTION_SORTS	total_section_sorts - Total section sorts
LOCK_ESCALS	lock_escals - Number of lock escalations
LOCK_WAITS	lock_waits - Lock waits
ROWS_MODIFIED	rows_modified - Rows modified
ROWS_READ	rows_read - Rows read
ROWS_RETURNED	rows_returned - Rows returned
DIRECT_READS	direct_reads - Direct reads from database
DIRECT_READ_REQS	direct_read_reqs - Direct read requests
DIRECT_WRITES	direct_writes - Direct writes to database
DIRECT_WRITE_REQS	direct_write_reqs - Direct write requests
POOL_DATA_L_READS	pool_data_l_reads - Buffer pool data logical reads
POOL_TEMP_DATA_L_READS	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_XDA_L_READS	pool_xda_l_reads - Buffer pool XDA data logical reads
POOL_TEMP_XDA_L_READS	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_INDEX_L_READS	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_INDEX_L_READS	pool_temp_index_l_reads - Buffer pool temporary index logical reads

Table 108. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing an activity_metrics element type (continued)

Metric Name	Description or Monitor element
POOL_DATA_P_READS	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_P_READS	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_XDA_P_READS	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_TEMP_INDEX_P_READS	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_INDEX_P_READS	pool_index_p_reads - Buffer pool index physical reads
POOL_DATA_WRITES	pool_data_writes - Buffer pool data writes
POOL_XDA_WRITES	pool_xda_writes - Buffer pool XDA data writes
POOL_INDEX_WRITES	pool_index_writes - Buffer pool index writes
TOTAL_SORTS	total_sorts - Total Sorts
POST_THRESHOLD_SORTS	post_threshold_sorts - Post threshold sorts
POST_SHRTHRESHOLD_SORTS	post_shrthreshold_sorts - Post shared threshold sorts
SORT_OVERFLOWS	sort_overflows - Sort overflows
WLM_QUEUE_ASSIGNMENTS_TOTAL	wlm_queue_assignments_total - Workload manager total queue assignments
DEADLOCKS	deadlocks - Deadlocks detected
FCM_RECV_VOLUME	fcm_recv_volume - FCM recv volume
FCM_RECVS_TOTAL	fcm_recvs_total - FCM recvs total
FCM_SEND_VOLUME	fcm_send_volume - FCM send volume
FCM_SENDS_TOTAL	fcm_sends_total - FCM sends total
LOCK_TIMEOUTS	lock_timeouts - Number of lock timeouts
NUM_LOG_BUFFER_FULL	num_log_buffer_full - Number of full log buffers
LOG_DISK_WAITS_TOTAL	log_disk_waits_total - Log disk waits total
TOTAL_ROUTINE_INVOCATIONS	total_routine_invocations - Total routine invocations
AUDIT_EVENTS_TOTAL	audit_events_total - Total audit events
AUDIT_SUBSYSTEM_WAITS_TOTAL	audit_subsystem_waits_total - Total audit subsystem waits
AUDIT_FILE_WRITES_TOTAL	audit_file_writes_total - Total Audit files written
DIAGLOG_WRITES_TOTAL	diaglog_writes_total - Diag log total writes
FCM_MESSAGE_RECV_VOLUME	fcm_message_recv_volume - FCM message recv volume
FCM_MESSAGE_RECVS_TOTAL	fcm_message_recvs_total - FCM message recvs total
FCM_MESSAGE_SEND_VOLUME	fcm_message_send_volume - FCM message send volume
FCM_MESSAGE_SENDS_TOTAL	fcm_message_sends_total - FCM message sends total
FCM_TQ_RECV_VOLUME	fcm_tq_recv_volume - FCM tablequeue recv volume
FCM_TQ_RECVS_TOTAL	fcm_tq_recvs_total - FCM tablequeue recvs total
FCM_TQ_SEND_VOLUME	fcm_tq_send_volume - FCM tablequeue send volume
FCM_TQ_SENDS_TOTAL	fcm_tq_sends_total - FCM tablequeue send total
TQ_TOT_SEND_SPILLS	tq_tot_send_spills - Total number of table queue buffers overflowed
THRESH_VIOLATIONS	thresh_violations - Number of threshold violations
NUM_LW_THRESH_EXCEEDED	num_lw_thresh_exceeded - Number of thresholds exceeded

Table 108. Metric names returned by MON_FORMAT_XML_METRICS_BY_ROW for XML documents containing an activity_metrics element type (continued)

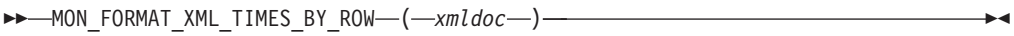
Metric Name	Description or Monitor element
COORD_STMT-EXEC_TIME	coord_stmt_exec_time - Execution time for statement by coordinator agent
TOTAL_ACT_TIME	total_act_time - Total activity time
TOTAL_CPU_TIME	total_cpu_time - Total CPU time

MON_FORMAT_XML_TIMES_BY_ROW - Get formatted row-based combined hierarchy wait and processing times

The MON_FORMAT_XML_TIMES_BY_ROW table function returns formatted row based output for the combined hierarchy of wait and processing times that are contained in an XML metrics document.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

Syntax



The schema is SYSPROC.

Table function parameters

xmldoc
An input argument of type BLOB(100M) that contains an XML document with either a system_metrics or activity_metrics element. XML documents with these elements can be obtained from the following sources:

- Returned by one of the MON_GET_*_DETAILS table functions.
- From the metrics column output by statistics and activity event monitors.
- From the formatted output of the unit of work, or package cache event monitors.

Authorization

EXECUTE privilege on the MON_FORMAT_XML_TIMES_BY_ROW function.

Example

To determine where time is being spent by your application within the DB2 database manger, you can run the following query to show the combined wait and processing time metrics in the metrics hierarchy.

```
SELECT SUBSTR(T.SERVICE_SUPERCLASS_NAME,1,15) as SUPERCLASS,
       SUBSTR(T.SERVICE_SUBCLASS_NAME,1,15) as SUBCLASS,
       T.MEMBER,
       SUBSTR(U.METRIC_NAME, 1,15) AS METRIC_NAME,
       SUBSTR(U.PARENT_METRIC_NAME,1,15) AS PARENT_NAME,
       U.TOTAL_TIME_VALUE,
```

```
       U.COUNT
FROM
TABLE(MON_GET_SERVICE_SUBCLASS_DETAILS(NULL, NULL, -2)) AS T,
TABLE(MON_FORMAT_XML_TIMES_BY_ROW(T.DETAILS)) AS U
```

The following is an example of output from this query.

```
SUPERCLASS SUBCLASS MEMBER METRIC_NAME     PARENT_NAME     T..._VALUE COUNT
---------- -------- ------ --------------- --------------- ---------- -----
MYSC       MYSSC         0 FCM_MESSAGE_REC FCM_RECV_WAIT_T          0     0
MYSC       MYSSC         0 FCM_TQ_RECV_WAI FCM_RECV_WAIT_T          0     0
MYSC       MYSSC         0 FCM_MESSAGE_SEN FCM_SEND_WAIT_T          0     0
MYSC       MYSSC         0 FCM_TQ_SEND_WAI FCM_SEND_WAIT_T          0     0
MYSC       MYSSC         0 TOTAL_COMMIT_PR TOTAL_RQST_TIME        300     1
MYSC       MYSSC         0 TOTAL_COMPILE_P TOTAL_RQST_TIME        700     1
MYSC       MYSSC         0 TOTAL_IMPLICIT_ TOTAL_RQST_TIME          0     0
MYSC       MYSSC         0 TOTAL_LOAD_PROC TOTAL_RQST_TIME          0     0
MYSC       MYSSC         0 TOTAL_REORG_PRO TOTAL_RQST_TIME          0     0
MYSC       MYSSC         0 TOTAL_ROLLBACK_ TOTAL_RQST_TIME          0     0
MYSC       MYSSC         0 TOTAL_RUNSTATS_ TOTAL_RQST_TIME          0     0
MYSC       MYSSC         0 TOTAL_SECTION_P TOTAL_RQST_TIME       7322     1
MYSC       MYSSC         0 TOTAL_WAIT_TIME TOTAL_RQST_TIME          0     0
MYSC       MYSSC         0 TOTAL_SECTION_S TOTAL_SECTION_P          0     0
MYSC       MYSSC         0 AGENT_WAIT_TIME TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 AUDIT_FILE_WRIT TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 AUDIT_SUBSYSTEM TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 DIAGLOG_WRITE_W TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 DIRECT_READ_TIM TOTAL_WAIT_TIME       1204    17
MYSC       MYSSC         0 DIRECT_WRITE_TI TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 FCM_RECV_WAIT_T TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 FCM_SEND_WAIT_T TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 IPC_RECV_WAIT_T TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 IPC_SEND_WAIT_T TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 LOCK_WAIT_TIME  TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 LOG_BUFFER_WAIT TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 LOG_DISK_WAIT_T TOTAL_WAIT_TIME        523     2
MYSC       MYSSC         0 POOL_READ_TIME  TOTAL_WAIT_TIME       2432     7
MYSC       MYSSC         0 POOL_WRITE_TIME TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 TCPIP_RECV_WAIT TOTAL_WAIT_TIME        523     1
MYSC       MYSSC         0 TCPIP_SEND_WAIT TOTAL_WAIT_TIME        241     1
MYSC       MYSSC         0 WLM_QUEUE_TIME_ TOTAL_WAIT_TIME          0     0
MYSC       MYSSC         0 CLIENT_IDLE_WAI -                      234     -
MYSC       MYSSC         0 TOTAL_RQST_TIME -                    13245     1
  34 record(s) selected.
```

Information returned

Table 109. Information returned for MON_FORMAT_XML_TIMES_BY_ROW

Column Name	Data Type	Description
METRIC_NAME	VARCHAR(128)	The unique identifier for the total time metric value.
TOTAL_TIME_VALUE	BIGINT	The total time value in milliseconds corresponding to metric_name.
COUNT	BIGINT	Number of occurrences of this type of interval.
PARENT_METRIC_NAME	VARCHAR(128)	The identifier of the parent total time metric whose value contains the total_time_value as a subset.

XML documents that contain an element of type *system_metrics* are generated from the following interfaces:

- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_UNIT_OF_WORK_DETAILS

- MON_GET_WORKLOAD_DETAILS
- DETAILS_XML column from a STATISTICS event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the UNIT OF WORK event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the UNIT OF WORK event monitor

See Table 110 for the types of metrics and their parent metrics that are returned from the XML document in this case:

Table 110. Metric names returned by MON_FORMAT_XML_TIMES_BY_ROW for XML documents containing a system_metrics element type

Metric Name	Parent Metric Name	Description of metric or Monitor element
TOTAL_RQST_TIME	NULL	total_rqst_time - Total request time
TOTAL_COMPILE_ PROC_TIME	TOTAL_RQST_TIME	total_compile_proc_time - Total compile processing time
TOTAL_IMPLICIT_ COMPILE_PROC_TIME	TOTAL_RQST_TIME	total_implicit_compile_proc_time - Total implicit compile processing time
TOTAL_SECTION_ PROC_TIME	TOTAL_RQST_TIME	total_section_proc_time - Total section processing time
TOTAL_COMMIT_ PROC_TIME	TOTAL_RQST_TIME	total_commit_proc_time - Total commits processing time
TOTAL_ROLLBACK _PROC_TIME	TOTAL_RQST_TIME	total_rollback_proc_time - Total rollback processing time
TOTAL_ROUTINE_USER _CODE_PROC_TIME	TOTAL_RQST_TIME	total_routine_user_code_proc_time - Total routine user code processing time
TOTAL_RUNSTATS_ PROC_TIME	TOTAL_RQST_TIME	total_runstats_proc_time - Total runtime statistics processing time
TOTAL_REORG _PROC_TIME	TOTAL_RQST_TIME	total_reorg_proc_time - Total reorganization processing time
TOTAL_LOAD_PROC_TIME	TOTAL_RQST_TIME	total_load_proc_time - Total load processing time
TOTAL_SECTION_ SORT_PROC_TIME	TOTAL_SECTION_ PROC_TIME	total_section_sort_proc_time - Total section sort processing time
TOTAL_WAIT_TIME	TOTAL_RQST_TIME	total_wait_time - Total wait time
CLIENT_IDLE_WAIT_TIME	NULL	client_idle_wait_time - Client idle wait time
POOL_READ_TIME	TOTAL_WAIT_TIME	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	TOTAL_WAIT_TIME	pool_write_time - Total buffer pool physical write time
DIRECT_READ_TIME	TOTAL_WAIT_TIME	direct_read_time - Direct read time
DIRECT_WRITE_TIME	TOTAL_WAIT_TIME	direct_write_time - Direct write time
LOCK_WAIT_TIME	TOTAL_WAIT_TIME	lock_wait_time - Time waited on locks
AGENT_WAIT_TIME	TOTAL_WAIT_TIME	agent_wait_time - Agent wait time
WLM_QUEUE_TIME_TOTAL	TOTAL_WAIT_TIME	wlm_queue_time_total - Workload manager total queue time
FCM_SEND_WAIT_TIME	TOTAL_WAIT_TIME	fcm_send_wait_time - FCM send wait time
FCM_RECV_WAIT_TIME	TOTAL_WAIT_TIME	fcm_recv_wait_time - FCM recv wait time
TCPIP_SEND_WAIT_TIME	TOTAL_WAIT_TIME	tcpip_send_wait_time - TCP/IP send wait time
TCPIP_RECV_WAIT_TIME	TOTAL_WAIT_TIME	tcpip_recv_wait_time - TCP/IP recv wait time

Table 110. Metric names returned by MON_FORMAT_XML_TIMES_BY_ROW for XML documents containing a system_metrics element type (continued)

Metric Name	Parent Metric Name	Description of metric or Monitor element
IPC_SEND_WAIT_TIME	TOTAL_WAIT_TIME	ipc_send_wait_time - Interprocess communication send wait time
IPC_RECV_WAIT_TIME	TOTAL_WAIT_TIME	ipc_recv_wait_time - Interprocess communication recv wait time
LOG_BUFFER_WAIT_TIME	TOTAL_WAIT_TIME	log_buffer_wait_time - Log buffer wait time
LOG_DISK_WAIT_TIME	TOTAL_WAIT_TIME	log_disk_wait_time - Log disk wait time
FCM_MESSAGE_ SEND_WAIT_TIME	FCM_SEND_WAIT_TIME	fcm_message_send_wait_time - FCM message send wait time
FCM_MESSAGE_ RECV_WAIT_TIME	FCM_RECV_WAIT_TIME	fcm_message_recv_wait_time - FCM message recv wait time
FCM_TQ_SEND_WAIT_TIME	FCM_SEND_WAIT_TIME	fcm_tq_send_wait_time - FCM tablequeue send wait time
FCM_TQ_RECV_WAIT_TIME	FCM_RECV_WAIT_TIME	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
AUDIT_FILE_WRITE _WAIT_TIME	TOTAL_WAIT_TIME	audit_file_write_wait_time - Audit file write wait time
AUDIT_SUBSYSTEM _WAIT_TIME	TOTAL_WAIT_TIME	audit_subsystem_wait_time - Audit subsystem wait time
DIAGLOG_WRITE_ WAIT_TIME	TOTAL_WAIT_TIME	diaglog_write_wait_time - Diag log write time

XML documents that contain an element of type *activity_metrics* are generated from the following interfaces:

- MON_GET_ACTIVITY_DETAILS
- MON_GET_PKG_CACHE_STMT_DETAILS
- DETAILS_XML column from an ACTIVITY event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the PACKAGE CACHE event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the PACKAGE CACHE event monitor

See Table 111 for the types of metrics and their parent metrics that are returned from the XML document in this case:

Table 111. Metric names returned by MON_FORMAT_XML_TIMES_BY_ROW for XML documents containing an activity_metrics element type

Metric Name	Parent Metric Name	Description or Monitor element
STMT_EXEC_TIME	NULL	stmt_exec_time - Statement execution time
TOTAL_ROUTINE_NON _SECT_PROC_TIME	STMT_EXEC_TIME	total_routine_non_sect_proc_time - Non-section processing time
TOTAL_ROUTINE_USER _CODE_PROC_TIME	TOTAL_ROUTINE_NON _SECT_PROC_TIME	total_routine_user_code_proc_time - Total routine user code processing time
TOTAL_SECTION_ PROC_TIME	STMT_EXEC_TIME	total_section_proc_time - Total section processing time
TOTAL_SECTION_SORT PROC_TIME	TOTAL_SECTION_ PROC_TIME	total_section_sort_proc_time - Total section sort processing time

Table 111. Metric names returned by MON_FORMAT_XML_TIMES_BY_ROW for XML documents containing an activity_metrics element type (continued)

Metric Name	Parent Metric Name	Description or Monitor element
TOTAL_ACT_WAIT_TIME	STMT_EXEC_TIME	total_act_wait_time - Total activity wait time
WLM_QUEUE_TIME_TOTAL	NULL	wlm_queue_time_total - Workload manager total queue time
POOL_READ_TIME	TOTAL_ACT_ WAIT_TIME	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	TOTAL_ACT_ WAIT_TIME	pool_write_time - Total buffer pool physical write time
DIRECT_READ_TIME	TOTAL_ACT_ WAIT_TIME	direct_read_time - Direct read time
DIRECT_WRITE_TIME	TOTAL_ACT_ WAIT_TIME	direct_write_time - Direct write time
LOCK_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	lock_wait_time - Time waited on locks
LOG_BUFFER_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	log_buffer_wait_time - Log buffer wait time
LOG_DISK_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	log_disk_wait_time - Log disk wait time
AUDIT_FILE_WRITE _WAIT_TIME	TOTAL_ACT_ WAIT_TIME	audit_file_write_wait_time - Audit file write wait time
AUDIT_SUBSYSTEM _WAIT_TIME	TOTAL_ACT_ WAIT_TIME	audit_subsystem_wait_time - Audit subsystem wait time
DIAGLOG_WRITE_ WAIT_TIME	TOTAL_ACT_ WAIT_TIME	diaglog_write_wait_time - Diag log write time
FCM_SEND_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	fcm_send_wait_time - FCM send wait time
FCM_RECV_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	fcm_recv_wait_time - FCM recv wait time
FCM_MESSAGE_ SEND_WAIT_TIME	FCM_SEND_ WAIT_TIME	fcm_message_send_wait_time - FCM message send wait time
FCM_MESSAGE_ RECV_WAIT_TIME	FCM_RECV_ WAIT_TIME	fcm_message_recv_wait_time - FCM message recv wait time
FCM_TQ_SEND_WAIT_TIME	FCM_SEND_ WAIT_TIME	fcm_tq_send_wait_time - FCM tablequeue send wait time
FCM_TQ_RECV_WAIT_TIME	FCM_RECV_ WAIT_TIME	fcm_tq_recv_wait_time - FCM tablequeue recv wait time

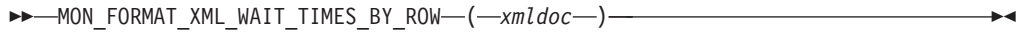
MON_FORMAT_XML_WAIT_TIMES_BY_ROW - Get formatted row-based output for wait times

The MON_FORMAT_XML_WAIT_TIMES_BY_ROW table function returns formatted row-based output for the wait times contained in an XML metrics document.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command

(because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

Syntax



The schema is SYSPROC.

Table function parameters

xmldoc
An input argument of type BLOB(100M) that contains an XML document with either a system_metrics or activity_metrics element. XML documents with these elements can be obtained from the following sources:

- Returned by one of the MON_GET_*_DETAILS table functions.
- From the metrics column output by statistics and activity event monitors.
- From the formatted output of the unit of work, or package cache event monitors.

Authorization

EXECUTE privilege on the MON_FORMAT_XML_WAIT_TIMES_BY_ROW function.

Example

This example shows how to call the MON_FORMAT_XML_WAIT_TIMES_BY_ROW table function to return formatted row-based output from the XML document produced by the MON_GET_WORKLOAD_DETAILS table function. The output shows the metrics and their values for each workload.

```
SELECT SUBSTR(TFXML.WORKLOAD_NAME, 1, 13) AS WORKLOAD_NAME,
       SUBSTR(WAITS.METRIC_NAME, 1, 25) AS METRIC_NAME,
       WAITS.TOTAL_TIME_VALUE,
       WAITS.COUNT
   FROM
     TABLE( MON_GET_WORKLOAD_DETAILS( NULL, -2 ) ) AS TFXML,
     TABLE( MON_FORMAT_XML_WAIT_TIMES_BY_ROW(
                                          TFXML.DETAILS
                                        )) AS WAITS
   ORDER BY WAITS.TOTAL_TIME_VALUE DESC
```

The following is a partial listing of the output of this query.

```
WORKLOAD_NAME METRIC_NAME               TOTAL_TIME_VALUE    COUNT
------------- ------------------------- ------------------- ------
PAYROLL       CLIENT_IDLE_WAIT_TIME                2193672    174
FINANCE       CLIENT_IDLE_WAIT_TIME                 738290     16
PAYROLL       DIRECT_READ_TIME                       67892     81
FINANCE       DIRECT_READ_TIME                       32343      8
FINANCE       LOCK_WAIT_TIME                          8463      3
PAYROLL       LOCK_WAIT_TIME                            55      1
```

Information returned

Table 112. Information returned for MON_FORMAT_XML_WAIT_TIMES_BY_ROW

Column Name	Data Type	Description
METRIC_NAME	VARCHAR(128)	The unique identifier for the total time metric value.
TOTAL_TIME_VALUE	BIGINT	The total time value in milliseconds corresponding to metric_name.
COUNT	BIGINT	Number of occurrences of this type of interval.
PARENT_METRIC_NAME	VARCHAR(128)	The identifier of the parent total time metric whose value contains the total_time_value as a subset.

XML documents that contain an element of type *system_metrics* are generated from the following interfaces:

- MON_GET_CONNECTION_DETAILS
- MON_GET_SERVICE_SUBCLASS_DETAILS
- MON_GET_UNIT_OF_WORK_DETAILS
- MON_GET_WORKLOAD_DETAILS
- DETAILS_XML column from a STATISTICS event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the UNIT OF WORK event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the UNIT OF WORK event monitor

See Table 113 for the types of metrics and their parent metrics that are returned from the XML document in this case:

Table 113. Metric names returned by MON_FORMAT_XML_WAIT_TIMES_BY_ROW for XML documents containing a system_metrics element type

Metric Name	Parent Metric Name	Description of metric or Monitor element
TOTAL_WAIT_TIME	TOTAL_RQST_TIME	total_wait_time - Total wait time
CLIENT_IDLE_WAIT_TIME	NULL	client_idle_wait_time - Client idle wait time
POOL_READ_TIME	TOTAL_WAIT_TIME	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	TOTAL_WAIT_TIME	pool_write_time - Total buffer pool physical write time
DIRECT_READ_TIME	TOTAL_WAIT_TIME	direct_read_time - Direct read time
DIRECT_WRITE_TIME	TOTAL_WAIT_TIME	direct_write_time - Direct write time
LOCK_WAIT_TIME	TOTAL_WAIT_TIME	lock_wait_time - Time waited on locks
AGENT_WAIT_TIME	TOTAL_WAIT_TIME	agent_wait_time - Agent wait time
WLM_QUEUE_TIME_TOTAL	TOTAL_WAIT_TIME	wlm_queue_time_total - Workload manager total queue time
FCM_SEND_WAIT_TIME	TOTAL_WAIT_TIME	fcm_send_wait_time - FCM send wait time
FCM_RECV_WAIT_TIME	TOTAL_WAIT_TIME	fcm_recv_wait_time - FCM recv wait time
TCPIP_SEND_WAIT_TIME	TOTAL_WAIT_TIME	tcpip_send_wait_time - TCP/IP send wait time
TCPIP_RECV_WAIT_TIME	TOTAL_WAIT_TIME	tcpip_recv_wait_time - TCP/IP recv wait time
IPC_SEND_WAIT_TIME	TOTAL_WAIT_TIME	ipc_send_wait_time - Interprocess communication send wait time
IPC_RECV_WAIT_TIME	TOTAL_WAIT_TIME	ipc_recv_wait_time - Interprocess communication recv wait time

Table 113. Metric names returned by MON_FORMAT_XML_WAIT_TIMES_BY_ROW for XML documents containing a system_metrics element type (continued)

Metric Name	Parent Metric Name	Description of metric or Monitor element
LOG_BUFFER_WAIT_TIME	TOTAL_WAIT_TIME	log_buffer_wait_time - Log buffer wait time
LOG_DISK_WAIT_TIME	TOTAL_WAIT_TIME	log_disk_wait_time - Log disk wait time
FCM_MESSAGE_ SEND_WAIT_TIME	FCM_SEND_ WAIT_TIME	fcm_message_send_wait_time - FCM message send wait time
FCM_MESSAGE_ RECV_WAIT_TIME	FCM_RECV_ WAIT_TIME	fcm_message_recv_wait_time - FCM message recv wait time
FCM_TQ_SEND_WAIT_TIME	FCM_SEND_ WAIT_TIME	fcm_tq_send_wait_time - FCM tablequeue send wait time
FCM_TQ_RECV_WAIT_TIME	FCM_RECV_ WAIT_TIME	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
AUDIT_FILE_WRITE _WAIT_TIME	TOTAL_WAIT_TIME	audit_file_write_wait_time - Audit file write wait time
AUDIT_SUBSYSTEM _WAIT_TIME	TOTAL_WAIT_TIME	audit_subsystem_wait_time - Audit subsystem wait time
DIAGLOG_WRITE_ WAIT_TIME	TOTAL_WAIT_TIME	diaglog_write_wait_time - Diag log write time

XML documents that contain an element of type *activity_metrics* are generated from the following interfaces:

- MON_GET_ACTIVITY_DETAILS
- MON_GET_PKG_CACHE_STMT_DETAILS
- DETAILS_XML column from an ACTIVITY event monitor
- METRICS column produced by EVMON_FORMAT_UE_TO_TABLES for the PACKAGE CACHE event monitor
- XMLREPORT column of EVMON_FORMAT_UE_TO_XML for the PACKAGE CACHE event monitor

See Table 114 for the types of metrics and their parent metrics that are returned from the XML document in this case:

Table 114. Metric names returned by MON_FORMAT_XML_WAIT_TIMES_BY_ROW for XML documents containing an activity_metrics element type

Metric Name	Parent Metric Name	Description or Monitor element
TOTAL_ACT_WAIT_TIME	STMT_EXEC_TIME	total_act_wait_time - Total activity wait time
WLM_QUEUE_TIME_TOTAL	NULL	wlm_queue_time_total - Workload manager total queue time
POOL_READ_TIME	TOTAL_ACT_ WAIT_TIME	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	TOTAL_ACT_ WAIT_TIME	pool_write_time - Total buffer pool physical write time
DIRECT_READ_TIME	TOTAL_ACT_ WAIT_TIME	direct_read_time - Direct read time
DIRECT_WRITE_TIME	TOTAL_ACT_ WAIT_TIME	direct_write_time - Direct write time
LOCK_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	lock_wait_time - Time waited on locks

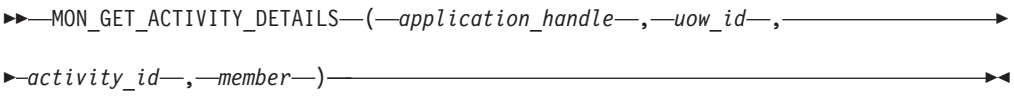
Table 114. Metric names returned by MON_FORMAT_XML_WAIT_TIMES_BY_ROW for XML documents containing an activity_metrics element type (continued)

Metric Name	Parent Metric Name	Description or Monitor element
LOG_BUFFER_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	log_buffer_wait_time - Log buffer wait time
LOG_DISK_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	log_disk_wait_time - Log disk wait time
AUDIT_FILE_WRITE _WAIT_TIME	TOTAL_ACT_ WAIT_TIME	audit_file_write_wait_time - Audit file write wait time
AUDIT_SUBSYSTEM _WAIT_TIME	TOTAL_ACT_ WAIT_TIME	audit_subsystem_wait_time - Audit subsystem wait time
DIAGLOG_WRITE_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	diaglog_write_wait_time - Diag log write time
FCM_SEND_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	fcm_send_wait_time - FCM send wait time
FCM_RECV_WAIT_TIME	TOTAL_ACT_ WAIT_TIME	fcm_recv_wait_time - FCM recv wait time
FCM_MESSAGE_ SEND_WAIT_TIME	FCM_SEND_ WAIT_TIME	fcm_message_send_wait_time - FCM message send wait time
FCM_MESSAGE_ RECV_WAIT_TIME	FCM_RECV_ WAIT_TIME	fcm_message_recv_wait_time - FCM message recv wait time
FCM_TQ_SEND_ WAIT_TIME	FCM_SEND_ WAIT_TIME	fcm_tq_send_wait_time - FCM tablequeue send wait time
FCM_TQ_RECV_ WAIT_TIME	FCM_RECV_ WAIT_TIME	fcm_tq_recv_wait_time - FCM tablequeue recv wait time

MON_GET_ACTIVITY_DETAILS table function - Get complete activity details

The MON_GET_ACTIVITY_DETAILS table function returns details about an activity, including general activity information (like statement text) and a set of metrics for the activity.

Syntax



The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies a valid application handle. If the argument is null, no rows are returned from this function, and an SQL0171N error is returned.

uow_id
An input argument of type INTEGER that specifies a valid unit of work

identifier unique within the application. If the argument is null, no rows are returned from this function, and an SQL0171N error is returned.

activity_id
An input argument of type INTEGER that specifies a valid activity ID unique within the unit of work. If the argument is null, no rows are returned from this function, and an SQL0171N error is returned.

member
An input argument of type INTEGER that specifies a valid member number in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_ACTIVITY_DETAILS function.

Example

Investigate a long running query to determine if it is spending its time executing or waiting (for example, blocked on locks or I/O).

Note: The following queries can be combined into one statement and are shown in 2 steps for reasons of clarity. Also, if you want to retrieve the complete text, you could use the executable ID to obtain the statement text from the MON_GET_PKG_CACHE_STMT table function.

1. First use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V9.7 table function to list activities and their start times.

```
SELECT application_handle,
       activity_id,
       uow_id,
       local_start_time
FROM TABLE(
   WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97(
      cast(NULL as bigint), -1)
) AS T
```

   The following is an example of output from this query.

```
APPLICATION_HANDLE   ACTIVITY_ID UOW_ID      LOCAL_START_TIME
-------------------- ----------- ----------- --------------------------
                   7           1           2 2008-06-10-10.06.55.675668
                  16           1           7 2008-06-10-10.08.38.613610

  2 record(s) selected.
```

2. Then use the MON_GET_ACTIVITY_DETAILS table function to view the percentage of time that the activity has spent waiting.

```
SELECT actmetrics.application_handle,
  actmetrics.activity_id,
  actmetrics.uow_id,
  varchar(actmetrics.stmt_text, 50) as stmt_text,
  actmetrics.total_act_time,
  actmetrics.total_act_wait_time,
  CASE WHEN actmetrics.total_act_time > 0
    THEN DEC((
      FLOAT(actmetrics.total_act_wait_time) /
      FLOAT(actmetrics.total_act_time)) * 100, 5, 2)
    ELSE NULL
  END AS PERCENTAGE_WAIT_TIME
FROM TABLE(MON_GET_ACTIVITY_DETAILS(7, 2, 1, -2)) AS ACTDETAILS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
```

```
  '$actmetrics/db2_activity_details'
  PASSING XMLPARSE(DOCUMENT ACTDETAILS.DETAILS) as "actmetrics"
  COLUMNS "APPLICATION_HANDLE" INTEGER PATH 'application_handle',
    "ACTIVITY_ID" INTEGER PATH 'activity_id',
    "UOW_ID" INTEGER PATH 'uow_id',
    "STMT_TEXT" VARCHAR(1024) PATH 'stmt_text',
    "TOTAL_ACT_TIME" INTEGER PATH 'activity_metrics/total_act_time',
    "TOTAL_ACT_WAIT_TIME" INTEGER PATH 'activity_metrics/total_act_wait_time'
  ) AS ACTMETRICS;
```

The following is an example of output from this query.

```
APPLICATION_HANDLE ACTIVITY_ID UOW_ID      ...
------------------ ----------- ----------- ...
                 7           1           2 ...

  1 record(s) selected.
```

Output for query (continued).

```
... STMT_TEXT                                          ...
... ------------------------------------------------- ...
... select * from syscat.tables optimize for 1 row     ...
```

Output for query (continued).

```
... TOTAL_ACT_TIME TOTAL_ACT_WAIT_TIME PERCENTAGE_WAIT_TIME
... -------------- -------------------- --------------------
...            459                    0                 0.00
```

Use the MON_GET_ACTIVITY_DETAILS table function to create a query that captures information about all the activities currently running on a system.

- Example 1: Run the following command using the DB2 command line processor (CLP)

```
WITH A1 AS
   (SELECT * FROM TABLE(wlm_get_workload_occurrence_activities_v97(null, -1))
    WHERE activity_id > 0 )
SELECT A1.application_handle,
  A1.activity_id,
  A1.uow_id,
  total_act_time,
  total_act_wait_time,
  varchar(actmetrics.stmt_text, 50) AS stmt_text FROM A1,
TABLE(MON_GET_ACTIVITY_DETAILS(A1.application_handle, A1.uow_id,A1.activity_id, -1))
    AS ACTDETAILS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
 '$actmetrics/db2_activity_details'
 PASSING XMLPARSE(DOCUMENT ACTDETAILS.DETAILS) AS "actmetrics"
 COLUMNS "STMT_TEXT" VARCHAR(1024) PATH 'stmt_text',
 "TOTAL_ACT_TIME" INTEGER PATH 'activity_metrics/total_act_time',
 "TOTAL_ACT_WAIT_TIME" INTEGER PATH 'activity_metrics/total_act_wait_time' )
 AS ACTMETRICS
```

The following is an example of output from this query:

```
APP...HANDLE  A..._ID  UOW_ID  T...ACT_TIME  T...WAIT_TIME
------------  -------  ------  ------------  -------------
15            1        5       16            5
15            1        3       17            5
7             1        49      0             0
SQL0445W Value "with A1 as (select * from table(wlm_get_workload
3 record(s) selected with 1 warning messages printed.
```

The following sample continues output from this query:

```
... STMT_TEXT
... -------------------------------------------------
... select name from sysibm.systables
... select * from sysibm.systables
... with A1 as (select * from table(wlm_get_workload_o
_occurrence_" has been truncated. SQLSTATE=01004

3 record(s) selected with 1 warning messages printed.
```

- Example 2 uses the FOR EACH ROW OF clause to imbed the function call to wlm_get_workload_occurrence_activities_v97 into the parameter list of the MON_GET_ACTIVITY_DETAILS function. Run the following command using the DB2 command line processor (CLP):

```
SELECT actmetrics.application_handle,
  actmetrics.activity_id,
  actmetrics.uow_id,
  varchar(actmetrics.stmt_text, 50) AS stmt_text,
  actmetrics.total_act_time,
  actmetrics.total_act_wait_time,
  CASE WHEN actmetrics.total_act_time > 0
    THEN DEC
      ((FLOAT(actmetrics.total_act_wait_time)/FLOAT(actmetrics.total_act_time)) * 100, 5, 2)
    ELSE NULL
  END AS PERCENTAGE_WAIT_TIME
FROM TABLE(MON_GET_ACTIVITY_DETAILS(for each row of (select application_handle,
  uow_id,
  activity_id from table(wlm_get_workload_occurrence_activities_v97(NULL, -1))
  WHERE activity_id > 0), -1)) AS ACTDETAILS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
  '$actmetrics/db2_activity_details'
  PASSING XMLPARSE(DOCUMENT ACTDETAILS.DETAILS) as "actmetrics"
  COLUMNS "APPLICATION_HANDLE" INTEGER PATH 'application_handle',
  "ACTIVITY_ID" INTEGER PATH 'activity_id',
  "UOW_ID" INTEGER PATH 'uow_id',
  "STMT_TEXT" VARCHAR(1024) PATH 'stmt_text',
  "TOTAL_ACT_TIME" INTEGER PATH 'activity_metrics/total_act_time',
  "TOTAL_ACT_WAIT_TIME" INTEGER
  PATH 'activity_metrics/total_act_wait_time' ) AS ACTMETRICS
```

The following is an example of output from this query:

```
APP...HANDLE  A..._ID  UOW_ID  STMT_TEXT                                          T...ACT_TIME  T...WT_TIME  P...WT_TIME
------------  -------  ------  -------------------------------------------------  ------------  ----------   -----------
93            1        8       SELECT actmetrics.application_handle, actmetrics.a  0             0            -
SQL0445W Value "SELECT actmetrics.application_handle, actmetrics.activity_id" has been truncated. SQLSTATE=01004
101           1        4       SELECT name FROM sysibm.systables                  12            0            0.00
101           1        3       SELECT 8 FROM sysibm.systables                     8             0            0.00

3 record(s) selected with 1 warning messages printed.
```

Usage notes

The MON_GET_ACTIVITY_DETAILS function provides maximum flexibility for formatting output because it returns detailed information for a single activity as an XML document. The XML output includes both descriptive information (for example, statement text) and metrics. The output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function as shown in the example.

The metrics reported through this function (for example, CPU usage) are rolled up to the activity periodically during the lifetime of the activity. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup.

Activity metrics are controlled through the COLLECT ACTIVITY METRICS clause on workloads, or the mon_act_metrics database configuration parameter at the database level. Metrics are collected if the connection that submits the activity is associated with a workload or database for which activity metrics are enabled. If activity metrics are not collected for an activity, all metrics are reported as 0.

The MON_GET_ACTIVITY_DETAILS table function returns one row of data for each member on which the activity exists. No aggregation across members is performed for the metrics. However, aggregation can be achieved through SQL queries.

The schema for the XML document that is returned in the DETAILS column is available in the file `sqllib/misc/DB2MonRoutines.xsd`. Further details can be found in the file `sqllib/misc/DB2MonCommon.xsd`.

Information returned

Table 115. Information returned for MON_GET_ACTIVITY_DETAILS

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
UOW_ID	INTEGER	uow_id - Unit of work ID
ACTIVITY_ID	INTEGER	activity_id - Activity ID
MEMBER	SMALLINT	member - Database member
DETAILS	BLOB(8M)	XML document that contains activity details. See Table 116 for a description of the elements in this document.

The following example shows the structure of the XML document that is returned in the DETAILS column.

```
<db2_activity_details xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="90700000">
  <member>0</member>
  <application_handle>70</application_handle>
  <activity_id>1</activity_id>
  <activity_state>IDLE</activity_state>
  <activity_type>READ_DML</activity_type>
  <uow_id>1</uow_id>
  ...
  <activity_metrics release="90700000">
    <lock_wait_time>2000</lock_wait_time>
    ...
  </activity_metrics>
</db2_activity_details>
```

For the full schema, see `sqllib/misc/DB2MonRoutines.xsd`. This document uses the following XML non-primitive type definitions:

```
<xs:simpleType name = "executable_id_type" >
  <xs:annotation>
     <xs:documentation>
       The binary Executable ID
     </xs:documentation>
  </xs:annotation>
   <xs:restriction base = "xs:hexBinary" >
      <xs:maxLength value = "32" />
   </xs:restriction>
</xs:simpleType>
```

Detailed metrics returned

Table 116. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS

Element name	Data type	Description or corresponding monitor element
member	xs:nonNegativeInteger	member - Database member
client_userid	xs:string(255)	CURRENT CLIENT_USERID special register

Table 116. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

Element name	Data type	Description or corresponding monitor element
client_wrkstnname	xs:string(255)	CURRENT CLIENT_WRKSTNNAME special register
client_applname	xs:string(255)	CURRENT CLIENT_APPLNAME special register
client_acctng	xs:string(255)	CURRENT CLIENT_ACCTNG special register
application_handle	xs:nonNegativeInteger	application_handle - Application handle
coord_member	xs:nonNegativeInteger	coord_member - Coordinator member
uow_id	xs:nonNegativeInteger	uow_id - Unit of work ID
activity_id	xs:nonNegativeInteger	activity_id - Activity ID
parent_uow_id	xs:nonNegativeInteger	parent_uow_id - Parent unit of work ID
parent_activity_id	xs:nonNegativeInteger	parent_activity_id - Parent activity ID
activity_state	xs:string	activity_state - Activity state
activity_type	xs:string	activity_type - Activity type
nesting_level	xs:nonNegativeInteger	stmt_nest_level - Statement nesting level
invocation_id	xs:nonNegativeInteger	stmt_invocation_id - Statement invocation identifier
routine_id	xs:nonNegativeInteger	routine_id - Routine ID
utility_id	xs:nonNegativeInteger	utility_id - Utility ID
service_class_id	xs:integer	service_class_id - Service class
database_work_action_set_id	xs:nonNegativeInteger	db_work_action_set_id - Database work action set ID
database_work_class_id	xs:nonNegativeInteger	db_work_class_id - Database work class ID
service_class_work _action_set_id	xs:nonNegativeInteger	sc_work_action_set_id - Service class work action set ID
service_class_work_class_id	xs:nonNegativeInteger	sc_work_class_id - Service class work class ID
entry_time	xs:dateTime	entry_time - Entry timeThe time that this activity arrived into the system.
local_start_time	xs:dateTime	local_start_time - Local start time.
last_reference_time	xs:dateTime	last_reference_time - Last reference time. Every time a request occurs in this activity, this field is updated.
package_name	xs:string (128)	package_name - Package name
package_schema	xs:string (128)	package_schema - Package schema
package_version_id	xs:string (128)	package_version_id - Package version
section_number	xs:integer	section_number - Section number
stmt_pkg_cache_id	xs:nonNegativeInteger	stmt_pkgcache_id - Statement package cache identifier
stmt_text	xs:string	stmt_text - SQL statement text. If the activity is dynamic SQL or it is static SQL for which the statement text is available, this field contains the first 1024 characters of the statement text. Otherwise, it contains an empty string.
effective_isolation	xs:string	effective_isolation - Effective isolation
effective_lock_timeout	xs:integer	effective_lock_timeout - Effective lock time-out
effective_query_degree	xs:integer	effective_query_degree - Effective query degree
query_cost_estimate	xs:integer	query_cost_estimate - Query cost estimate
qp_query_id	xs:nonNegativeInteger	qp_query_id - Query patroller query ID
num_remaps	xs:nonNegativeInteger	num_remaps - Number of remaps

Table 116. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

Element name	Data type	Description or corresponding monitor element
concurrentdbcoordactivities_db_threshold_id	xs:int	concurrentdbcoordactivities_db_threshold_id - Concurrent database coordinator activities threshold ID
concurrentdbcoordactivities_db_threshold_value	xs:long	concurrentdbcoordactivities_db_threshold_value - Concurrent database coordinator activities
concurrentdbcoordactivities_db_threshold_queued	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_db_threshold_queued - Concurrent database coordinator activities
concurrentdbcoordactivities_db_threshold_violated	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_db_threshold_violated - Concurrent database coordinator activities threshold violated
concurrentdbcoordactivities_superclass_threshold_id	xs:int	concurrentdbcoordactivities_superclass_threshold_id - Concurrent database coordinator activities superclass
concurrentdbcoordactivities_superclass_threshold_value	xs:long	concurrentdbcoordactivities_superclass_threshold_value - Concurrent database coordinator activities superclass threshold value
concurrentdbcoordactivities_superclass_threshold_queued	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_superclass_threshold_queued - Concurrent database coordinator activities superclass threshold queued
concurrentdbcoordactivities_superclass_threshold_violated	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_superclass_threshold_violated - Concurrent database coordinator activities superclass threshold violated
concurrentdbcoordactivities_subclass_threshold_id	xs:int	concurrentdbcoordactivities_subclass_threshold_id - Concurrent database coordinator activities subclass threshold ID
concurrentdbcoordactivities_subclass_threshold_value	xs:long	concurrentdbcoordactivities_subclass_threshold_value - Concurrent database coordinator activities subclass threshold value
concurrentdbcoordactivities_subclass_threshold_queued	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_subclass_threshold_queued - Concurrent database coordinator activities subclass threshold queued
concurrentdbcoordactivities_subclass_threshold_violated	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_subclass_threshold_violated - Concurrent database coordinator activities subclass threshold violated
concurrentdbcoordactivities_work_action_set_threshold_id	xs:int	concurrentdbcoordactivities_work_action_set_threshold_id - Concurrent database coordinator activities work action set threshold ID
concurrentdbcoordactivities_work_action_set_threshold_value	xs:long	concurrentdbcoordactivities_work_action_set_threshold_value - Concurrent database coordinator activities work action set threshold value
concurrentdbcoordactivities_work_action_set_threshold_queued	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_work_action_set_threshold_queued - Concurrent database coordinator activities work action set threshold queued
concurrentdbcoordactivities_work_action_set_threshold_violated	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_work_action_set_threshold_violated - Concurrent database coordinator activities work action set threshold violated
estimatedsqlcost_threshold_id	xs:int	estimatedsqlcost_threshold_id - Estimated SQL cost threshold ID
estimatedsqlcost_threshold_value	xs:long	estimatedsqlcost_threshold_value - Estimated SQL cost threshold value

Table 116. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

Element name	Data type	Description or corresponding monitor element
estimatedsqlcost_threshold _violated	xs:short (1 = yes, 0 = no)	estimatedsqlcost_threshold_violated - Estimated SQL cost threshold violated
sqltempspace_threshold_id	xs:int	sqltempspace_threshold_id - SQL temporary space threshold ID
sqltempspace_threshold _value	xs:long	sqltempspace_threshold_value - SQL temporary space threshold value
sqltempspace_threshold _violated	xs:short (1 = yes, 0 = no)	sqltempspace_threshold_violated - SQL temporary space threshold violated
sqlrowsreturned_threshold_id	xs:int	sqlrowsreturned_threshold_id - SQL rows read returned threshold ID
sqlrowsreturned_threshold _value	xs:long	sqlrowsreturned_threshold_value - SQL rows read returned threshold value
sqlrowsreturned_threshold _violated	xs:short (1 = yes, 0 = no)	sqlrowsreturned_threshold_violated - SQL rows read returned threshold violated
activitytotaltime_threshold_id	xs:int	activitytotaltime_threshold_id - Activity total time threshold ID
activitytotaltime_threshold _value	xs:dateTime	activitytotaltime_threshold_value - Activity total time threshold value
activitytotaltime_threshold _violated	xs:short (1 = yes, 0 = no)	activitytotaltime_threshold_violated - Activity total time threshold violated
cputime_threshold_id	xs:int	cputime_threshold_id - CPU time threshold ID
cputime_threshold_value	xs:long	cputime_threshold_value - CPU time threshold value
cputime_threshold_violated	xs:short (1 = yes, 0 = no)	cputime_threshold_violated - CPU time threshold violated
cputimeinsc_threshold_id	xs:int	cputimeinsc_threshold_id - CPU time in service threshold ID
cputimeinsc_threshold_value	xs:long	cputimeinsc_threshold_value - CPU time in service threshold value
cputimeinsc_threshold _violated	xs:short (1 = yes, 0 = no)	cputimeinsc_threshold_violated - CPU time in service threshold violated
sqlrowsread_threshold_id	xs:int	sqlrowsread_threshold_ID - SQL rows read threshold ID
sqlrowsread_threshold_value	xs:long	sqlrowsread_threshold_value - SQL rows read threshold value
sqlrowsread_threshold_violated	xs:short (1 = yes, 0 = no)	sqlrowsread_threshold_violated - SQL rows read threshold violated
sqlrowsreadinsc_threshold_id	xs:int	sqlrowsreadinsc_threshold_id - SQL rows read in service threshold ID
sqlrowsreadinsc_threshold _value	xs:long	sqlrowsreadinsc_threshold_value - SQL rows read in service threshold value
sqlrowsreadinsc_threshold _violated	xs:short (1 = yes, 0 = no)	sqlrowsreadinsc_threshold_violated - SQL rows read in service threshold violated
aggsqltempspace_threshold_id	xs:int	aggsqltempspace_threshold_id - AggSQL temporary space threshold ID.
aggsqltempspace_threshold _value	xs:long	aggsqltempspace_threshold_value - AggSQL temporary space threshold value
aggsqltempspace_threshold _violated	xs:short (1 = yes, 0 = no)	aggsqltempspace_threshold_violated - AggSQL temporary space threshold violated

Table 116. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

Element name	Data type	Description or corresponding monitor element
audit_events_total	xs:nonNegativeInteger	audit_events_total - Total audit events
audit_subsystem_wait_time	xs:nonNegativeInteger	audit_subsystem_wait_time - Audit subsystem wait time
audit_subsystem_waits_total	xs:nonNegativeInteger	audit_subsystem_waits_total - Total audit subsystem waits
audit_file_write_wait_time	xs:nonNegativeInteger	audit_file_write_wait_time - Audit file write wait time
audit_file_writes_total	xs:nonNegativeInteger	audit_file_writes_total - Total Audit files written
coord_stmt_exec_time		coord_stmt_exec_time - Execution time for statement by coordinator agent
deadlocks	xs:nonNegativeInteger	deadlocks - Deadlocks detected
diaglog_writes_total	xs:nonNegativeInteger	diaglog_writes_total - Diag log total writes
diaglog_write_wait_time	xs:nonNegativeInteger	diaglog_write_wait_time - Diag log write time
direct_read_time	xs:nonNegativeInteger	direct_read_time - Direct read time
direct_write_time	xs:nonNegativeInteger	direct_write_time - Direct write time
direct_read_reqs	xs:nonNegativeInteger	direct_read_reqs - Direct read requests
direct_reads	xs:nonNegativeInteger	direct_reads - Direct reads from database
direct_write_reqs	xs:nonNegativeInteger	direct_write_reqs - Direct write requests
direct_writes	xs:nonNegativeInteger	direct_writes - Direct writes to database
fcm_recv_volume	xs:nonNegativeInteger	fcm_recv_volume - FCM recv volume
fcm_recv_wait_time	xs:nonNegativeInteger	fcm_recv_wait_time - FCM recv wait time
fcm_recvs_total	xs:nonNegativeInteger	fcm_recvs_total - FCM recvs total
fcm_message_recv_volume	xs:nonNegativeInteger	fcm_message_recv_volume - FCM message recv volume
fcm_message_recvs_total	xs:nonNegativeInteger	fcm_message_recvs_total - FCM message recvs total
fcm_message_recv_wait_time	xs:nonNegativeInteger	fcm_message_recv_wait_time - FCM message recv wait time
fcm_message_send_volume	xs:nonNegativeInteger	fcm_message_send_volume - FCM message send volume
fcm_message_send_wait_time	xs:nonNegativeInteger	fcm_message_send_wait_time - FCM message send wait time
fcm_message_sends_total	xs:nonNegativeInteger	fcm_message_sends_total - FCM message sends total
fcm_send_volume	xs:nonNegativeInteger	fcm_send_volume - FCM send volume
fcm_send_wait_time	xs:nonNegativeInteger	fcm_send_wait_time - FCM send wait time
fcm_sends_total	xs:nonNegativeInteger	fcm_sends_total - FCM sends total
fcm_tq_recv_wait_time	xs:nonNegativeInteger	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
fcm_tq_send_wait_time	xs:nonNegativeInteger	fcm_tq_send_wait_time - FCM tablequeue send wait time
fcm_tq_recv_volume	xs:nonNegativeInteger	fcm_tq_recv_volume - FCM tablequeue recv volume
fcm_tq_recvs_total	xs:nonNegativeInteger	fcm_tq_recvs_total - FCM tablequeue recvs total
fcm_tq_send_volume	xs:nonNegativeInteger	fcm_tq_send_volume - FCM tablequeue send volume
fcm_tq_sends_total	xs:nonNegativeInteger	fcm_tq_sends_total - FCM tablequeue send total
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of tablequeue buffers overflowed
lock_escals	xs:nonNegativeInteger	lock_escals - Number of lock escalations
lock_timeouts	xs:nonNegativeInteger	lock_timeouts - Number of lock timeouts
lock_wait_time	xs:nonNegativeInteger	lock_wait_time - Time waited on locks

Table 116. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

Element name	Data type	Description or corresponding monitor element
lock_waits	xs:nonNegativeInteger	lock_waits - Lock waits
log_buffer_wait_time	xs:nonNegativeInteger	log_buffer_wait_time - Log buffer wait time
log_disk_wait_time	xs:nonNegativeInteger	log_disk_wait_time - Log disk wait time
log_disk_waits_total	xs:nonNegativeInteger	log_disk_waits_total - Log disk waits total
num_lw_thresh_exceeded	xs:nonNegativeInteger	num_lw_thresh_exceeded - Number of thresholds exceeded
pool_data_l_reads	xs:nonNegativeInteger	pool_data_l_reads - Buffer pool data logical reads
pool_data_p_reads	xs:nonNegativeInteger	pool_data_p_reads - Buffer pool data physical reads
pool_data_writes	xs:nonNegativeInteger	pool_data_writes - Buffer pool data writes
pool_index_l_reads	xs:nonNegativeInteger	pool_index_l_reads - Buffer pool index logical reads
pool_index_p_reads	xs:nonNegativeInteger	pool_index_p_reads - Buffer pool index physical reads
pool_index_writes	xs:nonNegativeInteger	pool_index_writes - Buffer pool index writes
pool_read_time	xs:nonNegativeInteger	pool_read_time - Total buffer pool physical read time
pool_temp_data_l_reads	xs:nonNegativeInteger	pool_temp_data_l_reads - Buffer pool temporary data logical reads
pool_temp_data_p_reads	xs:nonNegativeInteger	pool_temp_data_p_reads - Buffer pool temporary data physical reads
pool_temp_index_l_reads	xs:nonNegativeInteger	pool_temp_index_l_reads - Buffer pool temporary index logical reads
pool_temp_index_p_reads	xs:nonNegativeInteger	pool_temp_index_p_reads - Buffer pool temporary index physical reads
pool_temp_xda_l_reads	xs:nonNegativeInteger	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
pool_temp_xda_p_reads	xs:nonNegativeInteger	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
pool_write_time	xs:nonNegativeInteger	pool_write_time - Total buffer pool physical write time
pool_xda_l_reads	xs:nonNegativeInteger	pool_xda_l_reads - Buffer pool XDA data logical reads
pool_xda_p_reads	xs:nonNegativeInteger	pool_xda_p_reads - Buffer pool XDA data physical reads
pool_xda_writes	xs:nonNegativeInteger	pool_xda_writes - Buffer pool XDA data writes
num_log_buffer_full	xs:nonNegativeInteger	num_log_buffer_full - Number of full log buffers
rows_modified	xs:nonNegativeInteger	rows_modified - Rows modified
rows_read	xs:nonNegativeInteger	rows_read - Rows read
rows_returned	xs:nonNegativeInteger	rows_returned - Rows returned
stmt_exec_time	xs:nonNegativeInteger	stmt_exec_time - Statement execution time
thresh_violations	xs:nonNegativeInteger	thresh_violations - Number of threshold violations
total_cpu_time	xs:nonNegativeInteger	total_cpu_time - Total CPU time
total_act_time	xs:nonNegativeInteger	total_act_time - Total activity time
total_act_wait_time	xs:nonNegativeInteger	total_act_wait_time - Total activity wait time
total_app_section_executions	xs:nonNegativeInteger	total_app_section_executions - Total section executions
total_routine_invocations	xs:nonNegativeInteger	total_routine_invocations - Total routine invocations
total_routine_non_ sect_proc_time	xs:nonNegativeInteger	total_routine_non_sect_proc_time - Non-section processing time

Table 116. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

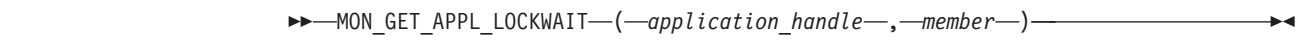
Element name	Data type	Description or corresponding monitor element
total_routine_non_ sect_time	xs:nonNegativeInteger	total_routine_non_sect_time - Non-section routine execution time
total_routine_time	xs:nonNegativeInteger	total_routine_time - Total routine time
total_routine_user_ code_proc_time	xs:nonNegativeInteger	total_routine_user_code_proc_time - Total routine user code processing time
total_routine_user_ code_time	xs:nonNegativeInteger	total_routine_user_code_time - Total routine user code time
total_section_proc_time	xs:nonNegativeInteger	total_section_proc_time - Total section processing time
total_section_sort_time	xs:nonNegativeInteger	total_section_sort_time - Total section sort time.
total_section_sort_ proc_time	xs:nonNegativeInteger	total_section_sort_proc_time - Total section sort processing time
total_section_sorts	xs:nonNegativeInteger	total_section_sorts - Total section sorts.
total_section_time	xs:nonNegativeInteger	total_section_time - Total section time
total_sorts	xs:nonNegativeInteger	total_sorts - Total Sorts
post_threshold_sorts	xs:nonNegativeInteger	post_threshold_sorts - Post threshold sorts
post_shrthreshold_sorts	xs:nonNegativeInteger	post_shrthreshold_sorts - Post shared threshold sorts
sort_overflows	xs:nonNegativeInteger	sort_overflows - Sort overflows
executable_id	executable_id_type	executable_id - Executable ID
wlm_queue_time_total	xs:nonNegativeInteger	wlm_queue_time_total - Workload manager total queue time
wlm_queue_assignments_total	xs:nonNegativeInteger	wlm_queue_assignments_total - Workload manager total queue assignments
eff_stmt_text	xs:string	eff_stmt_text - Effective statement text. The first 1024 characters of the concentrated statement text following any literal replacement done by the statement concentrator. Only present if the statement concentrator is enabled and this statement was altered by the statement concentrator.
wl_work_action_set_id	xs:nonNegativeInteger	wl_work_action_set_id - Workload work action set identifier monitor element
wl_work_class_id	xs:nonNegativeInteger	wl_work_class_id - Workload work class identifier
concurrentdbcoordactivities _wl_was_threshold_id	xs:int	concurrentdbcoordactivities_wl_was_threshold_id - Concurrent database coordinator activities workload work action set threshold ID
concurrentdbcoordactivities _wl_was_threshold_value	xs:long	concurrentdbcoordactivities_wl_was_threshold_value - Concurrent database coordinator activities workload work action set threshold value
concurrentdbcoordactivities _wl_was_threshold_queued	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_wl_was_threshold_queued - Concurrent database coordinator activities workload work action set threshold queued
concurrentdbcoordactivities _wl_was_threshold_violated	xs:short (1 = yes, 0 = no)	concurrentdbcoordactivities_wl_was_threshold_violated - Concurrent database coordinator activities workload work action set threshold violated

MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting

The MON_GET_APPL_LOCKWAIT table function returns information about all locks that each application's agents (that are connected to the current database) are waiting to acquire.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

To get information about locks, use the MON_GET_APPL_LOCKWAIT, MON_FORMAT_LOCK_NAME, and MON_GET_LOCKS, table functions instead of the SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function, and the SNAPLOCK administrative view and SNAP_GET_LOCK table function, which are deprecated in Fixpack 1 of Version 9.7.



The schema is SYSPROC.

Table function parameters

application_handle
An optional input parameter of type BIGINT that specifies a valid application handle in the same database as the one to which you are currently connected. If the argument is null, locks are retrieved for all applications that are currently waiting for locks to be acquired.

member
An input parameter of type INTEGER that specifies a valid member in the same instance as the currently connected database. Specify -1 for the current database member, or -2 for all active members. If the NULL value is specified, -1 is set.

Authorization

One of the following authorities or privilege is required:

- SYSADM authority
- SYSMON authority
- EXECUTE privilege on the MON_GET_APPL_LOCKWAIT table function.

Example

In this sample scenario, the MON_GET_APPL_LOCKWAIT table function is used to investigate a hung application for the session authorization ID USER1.

1. Use the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 table function to look up the application handle for all connections with the SESSION_USER value of USER1:

```
SELECT COORD_PARTITION_NUM,
     APPLICATION_HANDLE
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97('','',-2))
WHERE SESSION_USER = 'USER1'
```

This query returns the following output:

```
COORD_PARTITION_NUM              APPLICATION_HANDLE
-------------------------------  --------------------------
        2                        131130

1 record(s) selected.
```

2. Use the WLM_GET_SERVICE_CLASS_AGENTS_V97 table function to obtain current information on all agents working for this connection, on all database partitions:

```
SELECT SUBSTR(CHAR(DBPARTITIONNUM),1,3) AS DBPART,
     SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APP_ID,
     SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,7) AS WLO_ID,
     SUBSTR(CHAR(AGENT_TID),1,7) AS AGENT_ID,
     SUBSTR(CHAR(AGENT_TYPE),1,12) AS AGENT_TYPE,
     SUBSTR(AGENT_STATE,1, 8) AS STATE,
     SUBSTR(EVENT_TYPE,1, 8) AS EV_TYPE,
     SUBSTR(EVENT_OBJECT,1,12) AS EV_OBJECT
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS_V97('','',131130,-2))
ORDER BY AGENT_TYPE, DBPART
```

This query returns the following output:

```
DBPART  APP_ID  WLO_ID  AGENT_ID  AGENT_TYPE  STATE   EV_TYPE  EV_OBJECT
------  ------- ------- --------- ----------- ------  -------- ----------
2       131130  1       3110      COORDINATOR ACTIVE  WAIT      REQUEST
0       131130  1       7054      PDBSUBAGENT ACTIVE  ACQUIRE   LOCK
1       131130  1       5709      PDBSUBAGENT ACTIVE  ACQUIRE   LOCK
2       131130  1       5960      PDBSUBAGENT ACTIVE  ACQUIRE   LOCK

  4 record(s) selected.
```

An event of type ACQUIRE on an event object of type LOCK indicates a lock wait scenario, so we need to investigate which object is being waited for and who is holding the lock on it.

3. To determine all locks that the application is waiting for, call the MON_GET_APPL_LOCKWAIT table function with application handle 131130 and member -2 as input parameters.

```
SELECT lock_name,
       hld_member AS member,
       hld_agent_tid as TID,
       hld_application_handle AS HLD_APP FROM
       TABLE (MON_GET_APPL_LOCKWAIT(131130, -2))
```

This query returns the following output:

```
LOCK_NAME                  MEMBER TID    HLD_APP
-------------------------- ------ ------ -------
00030005000000000280000452    0   1234   65564
00030005000000000280000452    1   5478   65564
00030005000000000280000452    2   4678   65564

  3 record(s) selected.
```

4. Call the WLM_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 table function to find out more about the application that is holding the lock (this application has an application handle of 65564).

```
SELECT  SYSTEM_AUTH_ID,
   APPLICATION_NAME AS APP_NAME,
   WORKLOAD_NAME AS WORKLOAD,
   WORKLOAD_OCCURRENCE_STATE AS WL_STATE
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97('','',-2))
WHERE APPLICATION_HANDLE = 65564
```

This query returns the following output:

```
SYSTEM_AUTH_ID APP_NAME WORKLOAD                WL_STATE
-------------- -------- ---------------------- -----------
     ZURBIE    db2bp    SYSDEFAULTUSERWORKLOAD  UOWWAIT

  1 record(s) selected
```

Information returned

The columns that are returned provide information in the following areas:

- The following columns represent details about the lock that the application is currently waiting to acquire:

  LOCK_WAIT_START_TIME, LOCK_NAME, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_CURRENT_MODE, LOCK_MODE_REQUESTED, LOCK_STATUS, LOCK_ESCALATION, LOCK_ATTRIBUTES, LOCK_RRIID, LOCK_COUNT, TBSP_ID, TAB_FILE_ID, SUBSECTION_NUMBER.
- The following columns represent details about the application that is waiting to acquire this lock.

  REQ_APPLICATION_HANDLE, REQ_AGENT_TID, REQ_MEMBER, REQ_EXECUTABLE_ID
- The following columns represent details about the application that is currently holding the lock.

  HLD_APPLICATION_HANDLE, HLD_MEMBER, ADDITIONAL_DETAILS

Table 117. Information returned by the MON_GET_APPL_LOCKWAIT table function

Column name	Data type	Description or monitor element
LOCK_WAIT_START_TIME	TIMESTAMP	lock_wait_start_time - Lock Wait Start Timestamp
LOCK_NAME	VARCHAR(32)	lock_name - Lock name The internal name can be formatted using the MON_FORMAT_LOCK_NAME table function to obtain details regarding the lock. For example, the table and tablespace that the lock references can be found, if this is a table lock.
LOCK_OBJECT_TYPE	VARCHAR(32)	lock_object_type - Lock object type waited on For possible values, see "lock_object_type - Lock object type waited on monitor element"
LOCK_OBJECT_TYPE_ID	CHAR(1) FOR BIT DATA	Reserved for internal use
LOCK_MODE	VARCHAR(3)	lock_mode - Lock mode If the application holding this lock cannot be found, a value of NULL is returned.
LOCK_CURRENT_MODE	VARCHAR(3)	lock_current_mode - Original Lock Mode Before Conversion If no conversion took place, then a value of NULL is returned.

Table 117. Information returned by the MON_GET_APPL_LOCKWAIT table function (continued)

Column name	Data type	Description or monitor element
LOCK_MODE_REQUESTED	VARCHAR(3)	lock_mode_requested - Lock mode requested
LOCK_STATUS	CHAR(1)	lock_status - Lock status
LOCK_ESCALATION	CHAR(1)	lock_escalation - Lock escalation
LOCK_ATTRIBUTES	CHAR(16)	lock_attributes - Lock attributes
LOCK_RRIID	BIGINT	Reserved for internal use
LOCK_COUNT	BIGINT	Reserved for internal use
TBSP_ID	BIGINT	tablespace_id - Table space ID
TAB_FILE_ID	BIGINT	table_file_id - Table file ID
SUBSECTION_NUMBER	BIGINT	ss_number - Subsection Number If the subsection number is not available, then a value of NULL is returned.
REQ_APPLICATION_HANDLE	BIGINT	req_application_handle - Requesting application handle
REQ_AGENT_TID	BIGINT	req_agent_tid - Requesting agent TID
REQ_MEMBER	SMALLINT	req_member - Requesting member
REQ_EXECUTABLE_ID	VARCHAR (32)	req_executable_id - Requesting executable ID
HLD_APPLICATION_HANDLE	BIGINT	hld_application_handle - Holding application handle If the application holding this lock is unknown or cannot be found then a value of NULL is returned.
HLD_MEMBER	SMALLINT	hld_member - Holding member
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for internal use

MON_GET_BUFFERPOOL table function - Get buffer pool metrics

The MON_GET_BUFFERPOOL table function returns monitor metrics for one or more buffer pools.

Syntax

The schema is SYSPROC.

Table function parameters

bp_name
An input argument of type VARCHAR(128) that specifies a valid buffer pool

name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all buffer pools in the database.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_BUFFERPOOL function.

Example

Compute the buffer pool hit ratio.

```
WITH BPMETRICS AS (
    SELECT bp_name,
           pool_data_l_reads + pool_temp_data_l_reads +
           pool_index_l_reads + pool_temp_index_l_reads +
           pool_xda_l_reads + pool_temp_xda_l_reads as logical_reads,
           pool_data_p_reads + pool_temp_data_p_reads +
           pool_index_p_reads + pool_temp_index_p_reads +
           pool_xda_p_reads + pool_temp_xda_p_reads as physical_reads,
           member
    FROM TABLE(MON_GET_BUFFERPOOL('',-2)) AS METRICS)
   SELECT
    VARCHAR(bp_name,20) AS bp_name,
    logical_reads,
    physical_reads,
    CASE WHEN logical_reads > 0
     THEN DEC((1 - (FLOAT(physical_reads) / FLOAT(logical_reads))) * 100,5,2)
     ELSE NULL
    END AS HIT_RATIO,
    member
   FROM BPMETRICS;
```

The following is an example of output from this query.

```
BP_NAME          LOGICAL_READS    PHYSICAL_READS HIT_RATIO MEMBER
---------------- ---------------- -------------- --------- ---------
IBMDEFAULTBP                  619            385     37.80         0
IBMSYSTEMBP4K                   0              0         -         0
IBMSYSTEMBP8K                   0              0         -         0
IBMSYSTEMBP16K                  0              0         -         0
IBMSYSTEMBP32K                  0              0         -         0

  5 record(s) selected.
```

Output for query (continued).

```
... HIT_RATIO MEMBER
... --------- ------
...     37.80      0
...         -      0
...         -      0
...         -      0
...         -      0
```

Usage notes

The MON_GET_BUFFERPOOL table function returns one row of data per database buffer pool and per database member. No aggregation across database members is performed. However, aggregation can be achieved through SQL queries as shown in the example.

Metrics collected by this function are controlled at the database level using the mon_obj_metrics configuration parameter. By default, metrics collection is enabled.

Information returned

Table 118. Information returned for MON_GET_BUFFERPOOL

Column Name	Data Type	Description or corresponding monitor element
BP_NAME	VARCHAR(128)	
MEMBER	SMALLINT	member - Database member
AUTOMATIC	SMALLINT	automatic - Buffer pool automatic
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer pool XDA data logical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer pool XDA data writes
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time

Table 118. Information returned for MON_GET_BUFFERPOOL (continued)

Column Name	Data Type	Description or corresponding monitor element
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_ASYNC_DATA_READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_ASYNC_INDEX_READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer pool asynchronous XDA data reads
POOL_ASYNC_XDA_READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer pool asynchronous XDA read requests
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer pool asynchronous XDA data writes
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
POOL_LSN_GAP_CLNS	BIGINT	pool_lsn_gap_clns - Buffer pool log space cleaners triggered
POOL_DRTY_PG_STEAL_CLNS	BIGINT	pool_drty_pg_steal_clns - Buffer pool victim page cleaners triggered
POOL_DRTY_PG_THRSH_CLNS	BIGINT	pool_drty_pg_thrsh_clns - Buffer pool threshold cleaners triggered
VECTORED_IOS	BIGINT	vectored_ios - Number of vectored IO requests
PAGES_FROM_VECTORED_IOS	BIGINT	pages_from_vectored_ios - Total number of pages read by vectored IO
BLOCK_IOS	BIGINT	block_ios - Number of block IO requests
PAGES_FROM_BLOCK_IOS	BIGINT	pages_from_block_ios - Total number of pages read by block IO
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
FILES_CLOSED	BIGINT	files_closed - Database files closed
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use

MON_GET_CONNECTION table function - Get connection metrics

The MON_GET_CONNECTION table function returns metrics for one or more connections.

Syntax

The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies a specific application handle identifying the connection for which the metrics are to be returned. If the argument is null, metrics are returned for all connections

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_CONNECTION function.

Example

Display connections that return the highest volume of data to clients, ordered by rows returned.

```
SELECT application_handle,
       rows_returned,
       tcpip_send_volume
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
ORDER BY rows_returned DESC
```

The following is an example of output from this query.

```
APPLICATION_HANDLE   ROWS_RETURNED        TCPIP_SEND_VOLUME
-------------------- -------------------- --------------------
                  55                    6                    0

  1 record(s) selected.
```

Usage notes

The metrics returned by the MON_GET_CONNECTION table function represent the accumulation of all metrics for requests that were submitted by a connection. Metrics are rolled up at unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_CONNECTION table function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses and the *mon_req_metrics* database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default, request metrics are enabled at the database level. If request metrics are disabled at the database level and for a service superclass, then the metrics reported for each connection that is mapped to that service superclass will stop increasing (or remain at 0 if request metrics were disabled at database activation time).

Tip: As a connection can be mapped to more than one service superclass during its lifetime, the metrics reported through the MON_GET_CONNECTION table function might represent a subset of the metrics for all requests submitted over the connection. This might occur if metrics collection is disabled for some of the superclasses that are mapped by the connection.

The MON_GET_CONNECTION table function returns one row of data per connection and per member. No aggregation across members (for a service class or more), is performed. However, aggregation can be achieved through SQL queries.

Information returned

Table 119. Information returned for MON_GET_CONNECTION

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
APPLICATION_NAME	VARCHAR(128)	appl_name - Application name
APPLICATION_ID	VARCHAR(128)	appl_id - Application ID
MEMBER	SMALLINT	member - Database member
CLIENT_WRKSTNNAME	VARCHAR(255)	CURRENT CLIENT_WRKSTNNAME special register
CLIENT_ACCTNG	VARCHAR(255)	CURRENT CLIENT_ACCTNG special register
CLIENT_USERID	VARCHAR(255)	CURRENT CLIENT_USERID special register
CLIENT_APPLNAME	VARCHAR(255)	CURRENT CLIENT_APPLNAME special register
CLIENT_PID	BIGINT	client_pid - Client process ID
CLIENT_PRDID	VARCHAR(128)	client_prdid - Client product and version ID
CLIENT_PLATFORM	VARCHAR(12)	client_platform - Client platform
CLIENT_PROTOCOL	VARCHAR(10)	client_protocol - Client communication protocol
SYSTEM_AUTH_ID	VARCHAR(128)	system_auth_id - System authorization identifier
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
COORD_MEMBER	SMALLINT	coord_member - Coordinating member
CONNECTION_START_TIME	TIMESTAMP	connection_start_time - Connection start time
ACT_ABORTED_TOTAL	BIGINT	act_aborted_total - Total aborted activities
ACT_COMPLETED_TOTAL	BIGINT	act_completed_total - Total completed activities
ACT_REJECTED_TOTAL	BIGINT	act_rejected_total - Total rejected activities
AGENT_WAIT_TIME	BIGINT	agent_wait_time - Agent wait time
AGENT_WAITS_TOTAL	BIGINT	agent_waits_total - Total agent waits
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads

Table 119. Information returned for MON_GET_CONNECTION (continued)

Column name	Data type	Description
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer pool XDA data writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
CLIENT_IDLE_WAIT_TIME	BIGINT	client_idle_wait_time - Client idle wait time
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
FCM_RECV_VOLUME	BIGINT	fcm_recv_volume - FCM recv volume
FCM_RECVS_TOTAL	BIGINT	fcm_recvs_total - FCM recvs total
FCM_SEND_VOLUME	BIGINT	fcm_send_volume - FCM send volume
FCM_SENDS_TOTAL	BIGINT	fcm_sends_total - FCM sends total
FCM_RECV_WAIT_TIME	BIGINT	fcm_recv_wait_time - FCM recv wait time
FCM_SEND_WAIT_TIME	BIGINT	fcm_send_wait_time - FCM send wait time
IPC_RECV_VOLUME	BIGINT	ipc_recv_volume - Interprocess communication recv volume
IPC_RECV_WAIT_TIME	BIGINT	ipc_recv_wait_time - Interprocess communication recv wait time
IPC_RECVS_TOTAL	BIGINT	ipc_recvs_total - Interprocess communication recvs total
IPC_SEND_VOLUME	BIGINT	ipc_send_volume - Interprocess communication send volume
IPC_SEND_WAIT_TIME	BIGINT	ipc_send_wait_time - Interprocess communication send wait time
IPC_SENDS_TOTAL	BIGINT	ipc_sends_total - Interprocess communication send total
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations

Table 119. Information returned for MON_GET_CONNECTION (continued)

Column name	Data type	Description
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOG_BUFFER_WAIT_TIME	BIGINT	log_buffer_wait_time - Log buffer wait time
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers
LOG_DISK_WAIT_TIME	BIGINT	log_disk_wait_time - Log disk wait time
LOG_DISK_WAITS_TOTAL	BIGINT	log_disk_waits_total - Log disk waits total
NUM_LOCKS_HELD	BIGINT	locks_held - Locks held
RQSTS_COMPLETED_TOTAL	BIGINT	rqsts_completed_total - Total requests completed
ROWS_MODIFIED	BIGINT	rows_modified - Rows modified
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_RETURNED	BIGINT	rows_returned - Rows returned
TCPIP_RECV_VOLUME	BIGINT	tcpip_recv_volume - TCP/IP received volume
TCPIP_SEND_VOLUME	BIGINT	tcpip_send_volume - TCP/IP send volume
TCPIP_RECV_WAIT_TIME	BIGINT	tcpip_recv_wait_time - TCP/IP recv wait time
TCPIP_RECVS_TOTAL	BIGINT	tcpip_recvs_total - TCP/IP recvs total
TCPIP_SEND_WAIT_TIME	BIGINT	tcpip_send_wait_time - TCP/IP send wait time
TCPIP_SENDS_TOTAL	BIGINT	tcpip_sends_total - TCP/IP sends total
TOTAL_APP_RQST_TIME	BIGINT	total_app_rqst_time - Total application request time
TOTAL_RQST_TIME	BIGINT	total_rqst_time - Total request time
WLM_QUEUE_TIME_TOTAL	BIGINT	wlm_queue_time_total - Workload manager total queue time
WLM_QUEUE_ASSIGNMENTS_TOTAL	BIGINT	wlm_queue_assignments_total - Workload manager total queue assignments
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
TOTAL_WAIT_TIME	BIGINT	total_wait_time - Total wait time
APP_RQSTS_COMPLETED_TOTAL	BIGINT	app_rqsts_completed_total - Total application requests completed
TOTAL_SECTION_SORT_TIME	BIGINT	total_section_sort_time - Total section sort time
TOTAL_SECTION_SORT_PROC_TIME	BIGINT	total_section_sort_proc_time - Total section sort processing time
TOTAL_SECTION_SORTS	BIGINT	total_section_sorts - Total section sorts
TOTAL_SORTS	BIGINT	total_sorts - Total Sorts
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_COMPILE_TIME	BIGINT	total_compile_time - Total compile time
TOTAL_COMPILE_PROC_TIME	BIGINT	total_compile_proc_time - Total compile processing time
TOTAL_COMPILATIONS	BIGINT	total_compilations - Total compilations

Table 119. Information returned for MON_GET_CONNECTION (continued)

Column name	Data type	Description
TOTAL_IMPLICIT_COMPILE_TIME	BIGINT	total_implicit_compile_time - Total implicit compile time
TOTAL_IMPLICIT_COMPILE _PROC_TIME	BIGINT	total_implicit_compile_proc_time - Total implicit compile processing time
TOTAL_IMPLICIT_COMPILATIONS	BIGINT	total_implicit_compilations - Total implicit complications
TOTAL_SECTION_TIME	BIGINT	total_section_time - Total section time
TOTAL_SECTION_PROC_TIME	BIGINT	total_section_proc_time - Total section processing time
TOTAL_APP_SECTION_EXECUTIONS	BIGINT	total_app_section_executions - Total section executions
TOTAL_ACT_TIME	BIGINT	total_act_time - Total activity time
TOTAL_ACT_WAIT_TIME	BIGINT	total_act_wait_time - Total activity wait time
ACT_RQSTS_TOTAL	BIGINT	act_rqsts_total - Total activity requests
TOTAL_ROUTINE_TIME	BIGINT	total_routine_time - Total routine time
TOTAL_ROUTINE_INVOCATIONS	BIGINT	total_routine_invocations - Total routine invocations
TOTAL_COMMIT_TIME	BIGINT	total_commit_time - Total commit time
TOTAL_COMMIT_PROC_TIME	BIGINT	total_commit_proc_time - Total commits processing time
TOTAL_APP_COMMITS	BIGINT	total_app_commits - Total application commits
INT_COMMITS	BIGINT	int_commits - Internal commits
TOTAL_ROLLBACK_TIME	BIGINT	total_rollback_time - Total rollback time
TOTAL_ROLLBACK_PROC_TIME	BIGINT	total_rollback_proc_time - Total rollback processing time
TOTAL_APP_ROLLBACKS	BIGINT	total_app_rollbacks - Total application rollbacks
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
TOTAL_RUNSTATS_TIME	BIGINT	total_runstats_time - Total runtime statistics
TOTAL_RUNSTATS_PROC_TIME	BIGINT	total_runstats_proc_time - Total runtime statistics processing time
TOTAL_RUNSTATS	BIGINT	total_runstats - Total runtime statistics
TOTAL_REORG_TIME	BIGINT	total_reorg_time - Total reorganization time
TOTAL_REORG_PROC_TIME	BIGINT	total_reorg_proc_time - Total reorganization processing time
TOTAL_REORGS	BIGINT	total_reorgs - Total reorganizations
TOTAL_LOAD_TIME	BIGINT	total_load_time - Total load time
TOTAL_LOAD_PROC_TIME	BIGINT	total_load_proc_time - Total load processing time
TOTAL_LOADS	BIGINT	total_loads - Total loads
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
THRESH_VIOLATIONS	BIGINT	thresh_violations - Number of threshold violations

Table 119. Information returned for MON_GET_CONNECTION (continued)

Column name	Data type	Description
NUM_LW_THRESH_EXCEEDED	BIGINT	num_lw_thresh_exceeded - Number of thresholds exceeded
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use

MON_GET_CONNECTION_DETAILS table function - Get detailed connection metrics

The MON_GET_CONNECTION_DETAILS table function returns detailed metrics for one or more connections.

Syntax

The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies a specific application handle identifying the connection for which the metrics are to be returned. If the argument is NULL, metrics are returned for all connections.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_CONNECTION_DETAILS function.

Example

Display connections returning the highest volume of data to clients, ordered by rows returned.

```
SELECT detmetrics.application_handle,
       detmetrics.rows_returned,
       detmetrics.tcpip_send_volume
FROM TABLE(MON_GET_CONNECTION_DETAILS(CAST(NULL as bigint), -2))
AS CONNMETRICS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
       '$detmetric/db2_connection' PASSING XMLPARSE(DOCUMENT CONNMETRICS.DETAILS)
       as "detmetric"
COLUMNS "APPLICATION_HANDLE" INTEGER PATH 'application_handle',
       "ROWS_RETURNED" BIGINT PATH 'system_metrics/rows_returned',
       "TCPIP_SEND_VOLUME" BIGINT PATH 'system_metrics/tcpip_send_volume'
       ) AS DETMETRICS
ORDER BY rows_returned DESC
```

The following is an example of output from this query.

```
APPLICATION_HANDLE ROWS_RETURNED        TCPIP_SEND_VOLUME
------------------ -------------------- --------------------
                21                    4                    0

  1 record(s) selected.
```

Usage notes

The metrics returned by the MON_GET_CONNECTION_DETAILS table function represent the accumulation of all metrics for requests that were submitted by a connection. This function is similar to the MON_GET_CONNECTION table function:

- The MON_GET_CONNECTION table function returns the most commonly used metrics in a column-based format and is the most performance efficient method of retrieving metrics.
- The MON_GET_CONNECTION_DETAILS table function returns the entire set of available metrics in an XML document format, which provides maximum flexibility for formatting output. The XML-based output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function (see the example).

Metrics are rolled up at unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_CONNECTION_DETAILS table function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses and the **mon_req_metrics** database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default, request metrics are enabled are enabled at the database level. If request metrics are disabled at the database level, and for a service superclass, the metrics reported for each connection mapped to that service superclass stop increasing (or remain at 0 if request metrics were disabled at database activation time).

Tip: As a connection can be mapped to more than one service superclass during its lifetime, if collection is disabled at the database level, the metrics reported through the MON_GET_CONNECTION_DETAILS table function might represent a subset of the metrics for all requests submitted over the connection. This might occur if metrics collection is disabled for some of the superclasses to which the connection maps.

The MON_GET_CONNECTION_DETAILS table function returns one row of data per connection and per member. No aggregation across members (for a service class or more) is performed. However, aggregation can be achieved through SQL queries.

The schema for the XML document that is returned in the DETAILS column is available in the file `sqllib/misc/DB2MonRoutines.xsd`. Further details can be found in the file `sqllib/misc/DB2MonCommon.xsd`.

Information returned

Table 120. Information returned for MON_GET_CONNECTION_DETAILS

Column Name	Data Type	Description
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
MEMBER	SMALLINT	member - Database member
DETAILS	BLOB(1M)	XML document containing detailed metrics for the unit of work. See Table 121 for a description of the elements in this document.

The following example shows the structure of the XML document that is returned in the DETAILS column.

```
<db2_connection xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="90700000">
   <application_handle>21</application_handle>
   <member>0</member>
   <system_metrics release="90700000">
   <act_aborted_total>5</act_aborted_total>
   ...
   <wlm_queue_assignments_total>3</wlm_queue_assignments_total>
   </system_metrics>
</db2_connection>
```

For the full schema, see `sqllib/misc/DB2MonRoutines.xsd`.

This document uses the following non-primitive XML type definitions:

```
<xs:simpleType name="db2DbObjectString">
   <xs:restriction base="xs:string">
      <xs:maxLength value="128"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="db2PartitionNum">
   <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="999"/>
   </xs:restriction>
</xs:simpleType>
```

Table 121. Detailed metrics returned for MON_GET_CONNECTION_DETAILS

Element Name	Data Type	Description
act_aborted_total	xs:nonNegativeInteger	act_aborted_total - Total aborted activities
act_completed_total	xs:nonNegativeInteger	act_completed_total - Total completed activities
act_rejected_total	xs:nonNegativeInteger	act_rejected_total - Total rejected activities
act_rqsts_total	xs:nonNegativeInteger	act_rqsts_total - Total activity requests
agent_wait_time	xs:nonNegativeInteger	agent_wait_time - Agent wait time
agent_waits_total	xs:nonNegativeInteger	agent_waits_total - Total agent waits
application_handle	xs:nonNegativeInteger	application_handle - Application handle
application_id	xs:string	appl_id - Application ID
application_name	xs:string	appl_name - Application name
app_rqsts_completed_total	xs:nonNegativeInteger	app_rqsts_completed_total - Total application requests completed
audit_events_total	xs:nonNegativeInteger	audit_events_total - Total audit events
audit_subsystem_wait_time	xs:nonNegativeInteger	audit_subsystem_wait_time - Audit subsystem wait time

Table 121. Detailed metrics returned for MON_GET_CONNECTION_DETAILS (continued)

Element Name	Data Type	Description
audit_subsystem_waits_total	xs:nonNegativeInteger	audit_subsystem_waits_total - Total audit subsystem waits
audit_file_write_wait_time	xs:nonNegativeInteger	audit_file_write_wait_time - Audit file write wait time
audit_file_writes_total	xs:nonNegativeInteger	audit_file_writes_total - Total Audit files written
cat_cache_inserts	xs:nonNegativeInteger	cat_cache_inserts - Catalog cache inserts
cat_cache_lookups	xs:nonNegativeInteger	cat_cache_lookups - Catalog cache lookups
client_acctng	xs:string (255)	CURRENT CLIENT_ACCTNG special register
client_applname	xs:string (255)	CURRENT CLIENT_APPLNAME special register
client_hostname	xs:string	client_hostname - Client hostname
client_idle_wait_time	xs:nonNegativeInteger	client_idle_wait_time - Client idle wait time
client_pid	xs:nonNegativeInteger	client_pid - Client process ID
client_platform	xs:string	client_platform - Client platform
client_port_number	xs:nonNegativeInteger	client_port_number - Client port number
client_prdid	xs:string	client_prdid - Client product and version ID
client_protocol	xs:string	client_protocol - Client communication protocol
client_userid	xs:string (255)	CURRENT CLIENT_USERID special register
client_wrkstnname	xs:string (255)	CURRENT CLIENT_WRKSTNNAME special register
connection_start_time	xs:dateTime	connection_start_time - Connection start time
coord_member	xs:short	coord_member - Coordinating member
deadlocks	xs:nonNegativeInteger	deadlocks - Deadlocks detected
diaglog_writes_total	xs:nonNegativeInteger	diaglog_writes_total - Diag log total writes
diaglog_write_wait_time	xs:nonNegativeInteger	diaglog_write_wait_time - Diag log write time
direct_read_time	xs:nonNegativeInteger	direct_read_time - Direct read time
direct_write_time	xs:nonNegativeInteger	direct_write_time - Direct write time
direct_read_reqs	xs:nonNegativeInteger	direct_read_reqs - Direct read requests
direct_reads	xs:nonNegativeInteger	direct_reads - Direct reads from database
direct_write_reqs	xs:nonNegativeInteger	direct_write_reqs - Direct write requests
direct_writes	xs:nonNegativeInteger	direct_writes - Direct writes to database
fcm_recv_volume	xs:nonNegativeInteger	fcm_recv_volume - FCM recv volume
fcm_recv_wait_time	xs:nonNegativeInteger	fcm_recv_wait_time - FCM recv wait time
fcm_recvs_total	xs:nonNegativeInteger	fcm_recvs_total - FCM recvs total
fcm_message_recv_volume	xs:nonNegativeInteger	fcm_message_recv_volume - FCM message recv volume
fcm_message_recvs_total	xs:nonNegativeInteger	fcm_message_recvs_total - FCM message recvs total
fcm_message_recv_wait_time	xs:nonNegativeInteger	fcm_message_recv_wait_time - FCM message recv wait time
fcm_message_send_volume	xs:nonNegativeInteger	fcm_message_send_volume - FCM message send volume
fcm_message_send_wait_time	xs:nonNegativeInteger	fcm_message_send_wait_time - FCM message send wait time
fcm_message_sends_total	xs:nonNegativeInteger	fcm_message_sends_total - FCM message sends total

Table 121. Detailed metrics returned for MON_GET_CONNECTION_DETAILS (continued)

Element Name	Data Type	Description
fcm_send_volume	xs:nonNegativeInteger	fcm_send_volume - FCM send volume
fcm_send_wait_time	xs:nonNegativeInteger	fcm_send_wait_time - FCM send wait time
fcm_sends_total	xs:nonNegativeInteger	fcm_sends_total - FCM sends total
fcm_tq_recv_wait_time	xs:nonNegativeInteger	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
fcm_tq_send_wait_time	xs:nonNegativeInteger	fcm_tq_send_wait_time - FCM tablequeue send wait time
fcm_tq_recv_volume	xs:nonNegativeInteger	fcm_tq_recv_volume - FCM tablequeue recv volume
fcm_tq_recvs_total	xs:nonNegativeInteger	fcm_tq_recvs_total - FCM tablequeue recvs total
fcm_tq_send_volume	xs:nonNegativeInteger	fcm_tq_send_volume - FCM tablequeue send volume
fcm_tq_sends_total	xs:nonNegativeInteger	fcm_tq_sends_total - FCM tablequeue send total
int_commits	xs:nonNegativeInteger	int_commits - Internal commits
int_rollbacks	xs:nonNegativeInteger	int_rollbacks - Internal rollbacks
ipc_recv_volume	xs:nonNegativeInteger	ipc_recv_volume - Interprocess communication recv volume
ipc_recv_wait_time	xs:nonNegativeInteger	ipc_recv_wait_time - Interprocess communication recv wait time
ipc_recvs_total	xs:nonNegativeInteger	ipc_recvs_total - Interprocess communication recvs total
ipc_send_volume	xs:nonNegativeInteger	ipc_send_volume - Interprocess communication send volume
ipc_send_wait_time	xs:nonNegativeInteger	ipc_send_wait_time - Interprocess communication send wait time
ipc_sends_total	xs:nonNegativeInteger	ipc_sends_total - Interprocess communication send total
last_executable_id	xs:hexBinary(32)	last_executable_id - Last executable identifier
last_request_type	xs:string(32)	last_request_type - Last request type
lock_escals	xs:nonNegativeInteger	lock_escals - Number of lock escalations
lock_timeouts	xs:nonNegativeInteger	lock_timeouts - Number of lock timeouts
lock_wait_time	xs:nonNegativeInteger	lock_wait_time - Time waited on locks
lock_waits	xs:nonNegativeInteger	lock_waits - Lock waits
log_buffer_wait_time	xs:nonNegativeInteger	log_buffer_wait_time - Log buffer wait time
log_disk_wait_time	xs:nonNegativeInteger	log_disk_wait_time - Log disk wait time
log_disk_waits_total	xs:nonNegativeInteger	log_disk_waits_total - Log disk waits total
member	xs:nonNegativeInteger	member - Database member
num_locks_held	xs:nonNegativeInteger	locks_held - Locks held
num_log_buffer_full	xs:nonNegativeInteger	num_log_buffer_full - Number of full log buffers
num_lw_thresh_exceeded	xs:nonNegativeInteger	num_lw_thresh_exceeded - Number of thresholds exceeded
pkg_cache_inserts	xs:nonNegativeInteger	pkg_cache_inserts - Package cache inserts
pkg_cache_lookups	xs:nonNegativeInteger	pkg_cache_lookups - Package cache lookups
pool_data_l_reads	xs:nonNegativeInteger	pool_data_l_reads - Buffer pool data logical reads
pool_data_p_reads	xs:nonNegativeInteger	pool_data_p_reads - Buffer pool data physical reads
pool_data_writes	xs:nonNegativeInteger	pool_data_writes - Buffer pool data writes

Table 121. Detailed metrics returned for MON_GET_CONNECTION_DETAILS (continued)

Element Name	Data Type	Description
pool_index_l_reads	xs:nonNegativeInteger	pool_index_l_reads - Buffer pool index logical reads
pool_index_p_reads	xs:nonNegativeInteger	pool_index_p_reads - Buffer pool index physical reads
pool_index_writes	xs:nonNegativeInteger	pool_index_writes - Buffer pool index writes
pool_read_time	xs:nonNegativeInteger	pool_read_time - Total buffer pool physical read time
pool_temp_data_l_reads	xs:nonNegativeInteger	pool_temp_data_l_reads - Buffer pool temporary data logical reads
pool_temp_data_p_reads	xs:nonNegativeInteger	pool_temp_data_p_reads - Buffer pool temporary data physical reads
pool_temp_index_l_reads	xs:nonNegativeInteger	pool_temp_index_l_reads - Buffer pool temporary index logical reads
pool_temp_index_p_reads	xs:nonNegativeInteger	pool_temp_index_p_reads - Buffer pool temporary index physical reads
pool_temp_xda_l_reads	xs:nonNegativeInteger	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
pool_temp_xda_p_reads	xs:nonNegativeInteger	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
pool_write_time	xs:nonNegativeInteger	pool_write_time - Total buffer pool physical write time
pool_xda_l_reads	xs:nonNegativeInteger	pool_xda_l_reads - Buffer pool XDA data logical reads
pool_xda_p_reads	xs:nonNegativeInteger	pool_xda_p_reads - Buffer pool XDA data physical reads
pool_xda_writes	xs:nonNegativeInteger	pool_xda_writes - Buffer pool XDA data writes
post_shrthreshold_sorts	xs:nonNegativeInteger	post_shrthreshold_sorts - Post shared threshold sorts
post_threshold_sorts	xs:nonNegativeInteger	post_threshold_sorts - Post threshold sorts
rows_modified	xs:nonNegativeInteger	rows_modified - Rows modified
rows_read	xs:nonNegativeInteger	rows_read - Rows read
rows_returned	xs:nonNegativeInteger	rows_returned - Rows returned
rqsts_completed_total	xs:nonNegativeInteger	rqsts_completed_total - Total requests completed
session_auth_id	xs:string	session_auth_id - Session authorization ID
sort_overflows	xs:nonNegativeInteger	sort_overflows - Sort overflows
system_auth_id	xs:string	system_auth_id - System authorization identifier
tcpip_recv_volume	xs:nonNegativeInteger	tcpip_recv_volume - TCP/IP received volume
tcpip_recv_wait_time	xs:nonNegativeInteger	tcpip_recv_wait_time - TCP/IP recv wait time
tcpip_recvs_total	xs:nonNegativeInteger	tcpip_recvs_total - TCP/IP recvs total
tcpip_send_volume	xs:nonNegativeInteger	tcpip_send_volume - TCP/IP send volume
tcpip_send_wait_time	xs:nonNegativeInteger	tcpip_send_wait_time - TCP/IP send wait time
tcpip_sends_total	xs:nonNegativeInteger	tcpip_sends_total - TCP/IP sends total
thresh_violations	xs:nonNegativeInteger	num_thresh_violations - Number of threshold violations
total_act_time	xs:nonNegativeInteger	total_act_time - Total activity time
total_act_wait_time	xs:nonNegativeInteger	total_act_wait_time - Total activity wait time
total_app_commits	xs:nonNegativeInteger	total_app_commits - Total application commits
total_app_rollbacks	xs:nonNegativeInteger	total_app_rollbacks - Total application rollbacks

Table 121. Detailed metrics returned for MON_GET_CONNECTION_DETAILS (continued)

Element Name	Data Type	Description
total_app_rqst_time	xs:nonNegativeInteger	total_app_rqst_time - Total application request time
total_app_section_executions	xs:nonNegativeInteger	total_app_section_executions - Total section executions
total_commit_proc_time	xs:nonNegativeInteger	total_commit_proc_time - Total commits processing time
total_commit_time	xs:nonNegativeInteger	total_commit_time - Total commit time
total_compilations	xs:nonNegativeInteger	total_compilations - Total compilations
total_compile_proc_time	xs:nonNegativeInteger	total_compile_proc_time - Total compile processing time
total_compile_time	xs:nonNegativeInteger	total_compile_time - Total compile time
total_cpu_time	xs:nonNegativeInteger	total_cpu_time - Total CPU time
total_implicit_compilations	xs:nonNegativeInteger	total_implicit_compilations - Total implicit complications
total_implicit_compile_ proc_time	xs:nonNegativeInteger	total_implicit_compile_proc_time - Total implicit compile processing time
total_implicit_compile_time	xs:nonNegativeInteger	total_implicit_compile_time - Total implicit compile time
total_loads	xs:nonNegativeInteger	total_loads - Total loads
total_load_proc_time	xs:nonNegativeInteger	total_load_proc_time - Total load processing time
total_load_time	xs:nonNegativeInteger	total_load_time - Total load time
total_reorgs	xs:nonNegativeInteger	total_reorgs - Total reorganizations
total_reorg_proc_time	xs:nonNegativeInteger	total_reorg_proc_time - Total reorganization processing time
total_reorg_time	xs:nonNegativeInteger	total_reorg_time - Total reorganization time
total_rollback_proc_time	xs:nonNegativeInteger	total_rollback_proc_time - Total rollback processing time
total_rollback_time	xs:nonNegativeInteger	total_rollback_time - Total rollback time
total_routine_invocations	xs:nonNegativeInteger	total_routine_invocations - Total routine invocations
total_routine_user_code _proc_time	xs:nonNegativeInteger	total_routine_user_code_proc_time - Total routine user code processing time
total_routine_user_code_time	xs:nonNegativeInteger	total_routine_user_code_time - Total routine user code time
total_routine_time	xs:nonNegativeInteger	total_routine_time - Total routine time
total_rqst_time	xs:nonNegativeInteger	total_rqst_time - Total request time
total_runstats	xs:nonNegativeInteger	total_runstats - Total runtime statistics
total_runstats_proc_time	xs:nonNegativeInteger	total_runstats_proc_time - Total runtime statistics processing time
total_runstats_time	xs:nonNegativeInteger	total_runstats_time - Total runtime statistics
total_section_proc_time	xs:nonNegativeInteger	total_section_proc_time - Total section processing time
total_section_time	xs:nonNegativeInteger	total_section_time - Total section time
total_wait_time	xs:nonNegativeInteger	total_wait_time - Total wait time
total_section_sort_time	xs:nonNegativeInteger	total_section_sort_time - Total section sort time
total_section_sort_proc_time	xs:nonNegativeInteger	total_section_sort_proc_time - Total section sort processing time

Table 121. Detailed metrics returned for MON_GET_CONNECTION_DETAILS (continued)

Element Name	Data Type	Description
total_section_sorts	xs:nonNegativeInteger	total_section_sorts - Total section sorts
total_sorts	xs:nonNegativeInteger	total_sorts - Total Sorts
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of table queue buffers overflowed
wlm_queue_time_total	xs:nonNegativeInteger	wlm_queue_time_total - Workload manager total queue time
wlm_queue_assignments_total	xs:nonNegativeInteger	wlm_queue_assignments_total - Workload manager total queue assignments

MON_GET_CONTAINER table function - Get table space container metrics

The MON_GET_CONTAINER table function returns monitor metrics for one or more table space containers.

Syntax

The schema is SYSPROC.

Table function parameters

tbsp_name
An input argument of type VARCHAR(128) that specifies a valid table space name in the same database as the one currently connected to when calling this function. If the argument is null or an empty string, metrics are returned for all containers in all table spaces in the database.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_CONTAINER function.

Example

Example 1: List containers on all database members that have the highest read time.

```
SELECT varchar(container_name,70) as container_name,
       varchar(tbsp_name,20) as tbsp_name,
       pool_read_time
FROM TABLE(MON_GET_CONTAINER('',-2)) AS t
ORDER BY pool_read_time DESC
```

The following is an example of output from this query.

```
CONTAINER_NAME                                                               ...
---------------------------------------------------------------------------- ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000000/C0000000.CAT      ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000002/C0000000.LRG      ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000001/C0000000.TMP      ...

  3 record(s) selected.
```

Output for query (continued).

```
... TBSP_NAME            POOL_READ_TIME
... -------------------- --------------------
... SYSCATSPACE                         597
... USERSPACE1                           42
... TEMPSPACE1                            0
```

Example 2: List any containers that are not accessible.

```
SELECT varchar(container_name, 70) as container_name
FROM TABLE(MON_GET_CONTAINER('',-1)) AS t
WHERE accessible = 0
```

The following is an example of output from this query.

```
CONTAINER_NAME
----------------------------------------------------------------------

  0 record(s) selected.
```

Example 3: List utilization of container file systems, ordered by highest utilization.

```
SELECT varchar(container_name, 65) as container_name,
       fs_id,
       fs_used_size,
       fs_total_size,
       CASE WHEN fs_total_size > 0
            THEN DEC(100*(FLOAT(fs_used_size)/FLOAT(fs_total_size)),5,2)
            ELSE DEC(-1,5,2)
       END as utilization
FROM TABLE(MON_GET_CONTAINER('',-1)) AS t
ORDER BY utilization DESC
```

The following is an example of output from this query.

```
CONTAINER_NAME                                                         ...
---------------------------------------------------------------------- ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000000/C0000000.CAT ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000001/C0000000.TMP ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000002/C0000000.LRG ...

  3 record(s) selected.
```

Output for query (continued).

```
FS_ID                FS_USED_SIZE         FS_TOTAL_SIZE        UTILIZATION
-------------------- -------------------- -------------------- -----------
               64768         106879311872         317068410880       33.70
               64768         106879311872         317068410880       33.70
               64768         106879311872         317068410880       33.70
```

Usage notes

The MON_GET_CONTAINER table function returns one row of data per container and per database member. Data can be returned for all containers in a given table space, or for all containers in the database. No aggregation across database partitions is performed. However, aggregation can be achieved through SQL queries.

Metrics collected by this function are controlled at the database level using the mon_obj_metrics configuration parameter. By default, metrics collection is enabled.

Information returned

Table 122. Information returned for MON_GET_CONTAINER

Column Name	Data Type	Description or corresponding monitor element
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
CONTAINER_NAME	VARCHAR(256)	container_name - Container name
CONTAINER_ID	BIGINT	container_id - Container identification
MEMBER	SMALLINT	member - Database member
CONTAINER_TYPE	VARCHAR(16)	container_type - Container type This is a text identifier based on the defines in sqlutil.h and is one of: • DISK_EXTENT_TAG • DISK_PAGE_TAG • FILE_EXTENT_TAG • FILE_PAGE_TAG • PATH
STRIPE_SET	BIGINT	container_stripe_set - Stripe set
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
PAGES_READ	BIGINT	pages_read - Number of pages read
PAGES_WRITTEN	BIGINT	pages_written - Number of pages written
VECTORED_IOS	BIGINT	vectored_ios - Number of vectored IO requests
PAGES_FROM_VECTORED_IOS	BIGINT	pages_from_vectored_ios - Total number of pages read by vectored IO
BLOCK_IOS	BIGINT	block_ios - Number of block IO requests
PAGES_FROM_BLOCK_IOS	BIGINT	pages_from_block_ios - Total number of pages read by block IO
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
TOTAL_PAGES	BIGINT	container_total_pages - Total pages in container
USABLE_PAGES	BIGINT	container_usable_pages - Usable pages in container
ACCESSIBLE	SMALLINT	container_accessible - Accessibility of container
FS_ID	VARCHAR(22)	fs_id - Unique file system identification number
FS_TOTAL_SIZE	BIGINT	fs_total_size - Total size of a file system
FS_USED_SIZE	BIGINT	fs_used_size - Amount of space used on a file system
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

MON_GET_EXTENT_MOVEMENT_STATUS - get extent movement progress

The MON_GET_EXTENT_MOVEMENT_STATUS table function returns the status of the extent movement operation.

Syntax

The schema is SYSPROC.

Table function parameters

tbsp_name
An input argument of type VARCHAR(128) that specifies the table space to query. If the argument value is null, the function returns information for all table spaces.

member
An input argument of type INTEGER that specifies a valid member inside the same instance as the currently connected database. Specify -1 for the current database member, or -2 for all database members. If the argument value is null, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_EXTENT_MOVEMENT_STATUS function.

Example

Retrieve all information on the current extent progress for all table spaces:

```
SELECT * FROM TABLE(SYSPROC.MON_GET_EXTENT_MOVEMENT_STATUS('', -1))
```

Here is an example of the output from the above query:

```
TBSP_NAME   TBSP_ID MEMBER CURRENT_EXTENT LAST_EXTENT NUM_EXTENTS_MOVED
---------   ------- ------ -------------- ----------- -----------------
SYSCATSPACE       0      0             -1          -1                -1
TEMPSPACE1        1      0             -1          -1                -1
USERSPACE1        2      0             -1          -1                -1
TS1               3      0              1           2                 3
SYSTOOLSPACE      4      0             -1          -1                -1

5 record(s) selected.
```

Output from the query continued:

```
... NUM_EXTENTS_LEFT TOTAL_MOVE_TIME ADDITIONAL_DETAILS
... ---------------- --------------- ------------------
...               -1              -1                  -
...               -1              -1                  -
...               -1              -1                  -
...                4               0                  -
...               -1              -1                  -
```

Information returned

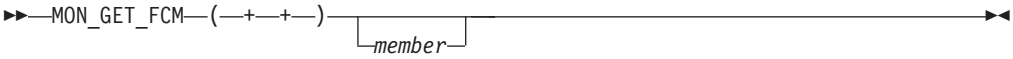
Table 123. Information returned for MON_GET_EXTENT_MOVEMENT_STATUS

Column Name	Data Type	Description or corresponding monitor element
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identifier
MEMBER	SMALLINT	member - Member from which this information was collected
CURRENT_EXTENT	INTEGER	current_extent - Current extent being moved
LAST_EXTENT	INTEGER	last_extent - Last extent moved
NUM_EXTENTS_MOVED	INTEGER	num_extents_moved - Number of extents moved so far during this extent movement operation
NUM_EXTENTS_LEFT	INTEGER	num_extents_left - Number of extents left to move during this extent movement operation
TOTAL_MOVE_TIME	BIGINT	total_move_time - Total move time for all extents moved (in milliseconds)
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use

MON_GET_FCM - Get FCM metrics

The MON_GET_FCM table function returns metrics for the fast communication manager (FCM).

Syntax



The schema is SYSPROC.

Table function parameter

member
An input argument of type INTEGER that specifies a valid database member number. Specify -1 for the current database member, or -2 for information from all active database members. An active database member is where the database is available for connection and use by applications.

Authorization

EXECUTE privilege on the MON_GET_FCM table function.

Information returned

Table 124. Information returned for MON_GET_FCM

Column Name	Data Type	Description or corresponding monitor element
HOSTNAME	VARCHAR(128)	hostname - Host name

Table 124. Information returned for MON_GET_FCM (continued)

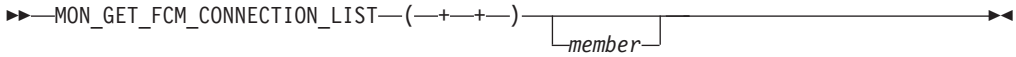
Column Name	Data Type	Description or corresponding monitor element
MEMBER	SMALLINT	member - Database member
BUFF_MAX	BIGINT	buff_max - Maximum possible number of FCM buffers
BUFF_TOTAL	BIGINT	buff_total - Number of currently allocated FCM buffers
BUFF_FREE	BIGINT	buff_free - FCM buffers currently free
BUFF_FREE_BOTTOM	BIGINT	buff_free_bottom - Minimum FCM buffers free
BUFF_AUTO_TUNING	SMALLINT	buff_auto_tuning - FCM buffer auto-tuning indicator
CH_MAX	BIGINT	ch_max - Maximum possible number of FCM channels
CH_TOTAL	BIGINT	ch_total - Number of currently allocated FCM channels
CH_FREE	BIGINT	ch_free - Channels currently free
CH_FREE_BOTTOM	BIGINT	ch_free_bottom - Minimum channels free
CH_AUTO_TUNING	SMALLINT	ch_auto_tuning - FCM channel auto-tuning indicator
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

Note: The metrics provided by this table function apply to all members on a given host machine. All members on a given host machine share the same set of buffers and channels. This means that the individual metrics will be the same for each member on given host machine.

MON_GET_FCM_CONNECTION_LIST - Get details for all FCM connections

The MON_GET_FCM_CONNECTION_LIST table function returns monitor metrics for all the fast communication manager (FCM) connections on the specified member or members.

Syntax



The schema is SYSPROC.

Table function parameter

member
An input argument of type INTEGER that specifies a valid database member number. Specify -1 for the current database member, or -2 for information from all active database members. An active database member is where the database is available for connection and use by applications.

Authorization

EXECUTE privilege on the MON_GET_FCM_CONNECTION_LIST table function.

Information returned

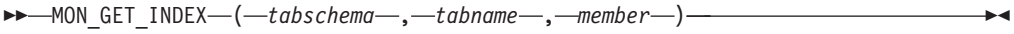
Table 125. Information returned for MON_GET_FCM_CONNECTION_LIST

Column Name	Data Type	Description or corresponding monitor element
MEMBER	SMALLINT	member - Database member
REMOTE_MEMBER	SMALLINT	remote_member - Remote member
CONNECTION_STATUS	VARCHAR(16)	connection_status - Connection status
TOTAL_BUFFERS_SENT	BIGINT	total_buffers_sent - Total FCM buffers sent
TOTAL_BUFFERS_RCVD	BIGINT	total_buffers_rcvd - Total FCM buffers received
FCM_CONGESTION_TIME	BIGINT	Reserved for future use.
FCM_CONGESTED_SENDS	BIGINT	Reserved for future use.
FCM_NUM_CONGESTION_TIMEOUTS	BIGINT	Reserved for future use.
FCM_SEND_VOLUME	BIGINT	Reserved for future use.
FCM_RECV_VOLUME	BIGINT	Reserved for future use.
FCM_MESSAGE_SEND_VOLUME	BIGINT	Reserved for future use.
FCM_MESSAGE_RECV_VOLUME	BIGINT	Reserved for future use.
FCM_TQ_SEND_VOLUME	BIGINT	Reserved for future use.
FCM_TQ_RECV_VOLUME	BIGINT	Reserved for future use.
FCM_NUM_CONN_LOST	BIGINT	Reserved for future use.
FCM_NUM_CONN_TIMEOUTS	BIGINT	Reserved for future use.
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

MON_GET_INDEX table function - get index metrics

The MON_GET_INDEX table function returns metrics for one or more indexes.

Syntax



The schema is SYSPROC.

Table function parameters

tabschema
An input argument of type VARCHAR(128) that specifies a valid table schema name in the same database as the one currently connected to when calling this function. If the argument is NULL or an empty string, metrics are retrieved for indexes of tables in all schemas in the database. If the argument is specified, metrics are only returned for indexes for tables in the specified schema.

tabname
An input argument of type VARCHAR(128) that specifies a valid table name in the same database as the one currently connected to when calling this function. Metrics are returned for all indexes on the specified table. If the argument is null or an empty string, metrics are retrieved for all indexes for all tables in the database.

member
An input argument of type INTEGER that specifies a valid member in the

same instance as the currently connected database when calling this function. Specify a -1 for the current database member, or -2 for all database members. If the NULL value is specified, -1 is set implicitly

Authorization

EXECUTE privilege on the MON_GET_INDEX function.

Example

Identify the most frequently used indexes on the DMEXT002.TABLE1 table, since the last database activation:

```
SELECT VARCHAR(S.INDSCHEMA, 10) AS INDSCHEMA,
  VARCHAR(S.INDNAME, 10) AS INDNAME,
  T.DATA_PARTITION_ID,
  T.MEMBER,
  T.INDEX_SCANS,
  T.INDEX_ONLY_SCANS
FROM TABLE(MON_GET_INDEX('DMEXT002','TABLE1', -2)) as T, SYSCAT.INDEXES AS S
WHERE T.TABSCHEMA = S.TABSCHEMA AND
  T.TABNAME = S.TABNAME AND
  T.IID = S.IID
ORDER BY INDEX_SCANS DESC
```

The following is an example of output from this query.

```
INDSCHEMA  INDNAME    DATA_PARTITION_ID MEMBER         INDEX_SCANS          INDEX_ONLY_SCANS
---------- ---------- ----------------- -------------- -------------------- --------------------
DMEXT002   INDEX3                    -              0                    1                    1
DMEXT002   INDEX4                    -              0                    1                    0
DMEXT002   INDEX1                    -              0                    0                    0
DMEXT002   INDEX2                    -              0                    0                    0
DMEXT002   INDEX5                    -              0                    0                    0
DMEXT002   INDEX6                    -              0                    0                    0

  6 record(s) selected.
```

Usage notes

The MON_GET_INDEX table function returns one row of data per index, and per database member. If partitioned indexes are being used, one row is returned for each index partition per database member. No aggregation across database members is performed. However, aggregation can be achieved through SQL queries as shown in the example above.

Metrics will only be returned for indexes on tables that have been accessed since the database was activated. All counters represent data since the current database activation. For example, the *pseudo_empty_pages* counter is the number of pages that have been identified as pseudo empty since the database was activated. it is not the current number of pseudo empty pages in the index.

Metrics are always enabled. It is not necessary to turn on any system monitor switches to access table metrics through this function.

Information returned

Table 126. Information returned for MON_GET_INDEX

Column Name	Data Type	Description or corresponding monitor element
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name

Table 126. Information returned for MON_GET_INDEX (continued)

Column Name	Data Type	Description or corresponding monitor element
IID	SMALLINT	iid - Index identifier
MEMBER	SMALLINT	member - Database member
DATA_PARTITION_ID	INTEGER	data_partition_id - Data partition identifier. If index is not partitioned, NULL is returned.
NLEAF	BIGINT	nleaf - Number of leaf pages
NLEVELS	SMALLINT	nlevels - Number of index levels
INDEX_SCANS	BIGINT	index_scans - Index scans
INDEX_ONLY_SCANS	BIGINT	index_only_scans - Index-only scans
KEY_UPDATES	BIGINT	key_updates - Key updates
INCLUDE_COL_UPDATES	BIGINT	include_col_updates - Include column updates
PSEUDO_DELETES	BIGINT	pseudo_deletes - Pseudo deletes
DEL_KEYS_CLEANED	BIGINT	del_keys_cleaned - Pseudo deleted keys cleaned
ROOT_NODE_SPLITS	BIGINT	root_node_splits - Root node splits
INT_NODE_SPLITS	BIGINT	int_node_splits - Intermediate node splits
BOUNDARY_LEAF _NODE_SPLITS	BIGINT	boundary_leaf_node_splits - Boundary leaf node splits
NONBOUNDARY_LEAF _NODE_SPLITS	BIGINT	nonboundary_leaf_node_splits - Non-boundary leaf node splits
PAGE_ALLOCATIONS	BIGINT	page_allocations - Page allocations
PSEUDO_EMPTY_PAGES	BIGINT	pseudo_empty_pages - Pseudo empty pages
EMPTY_PAGES_REUSED	BIGINT	empty_pages_reused - Empty pages reused
EMPTY_PAGES_DELETED	BIGINT	empty_pages_deleted - Empty pages deleted
PAGES_MERGED	BIGINT	pages_merged - Pages merged
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

MON_GET_LOCKS - list all locks in the currently connected database

The MON_GET_LOCKS table function returns a list of all locks in the currently connected database.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

To get information about locks, use the MON_GET_LOCKS, MON_FORMAT_LOCK_NAME, and MON_GET_APPL_LOCKWAIT table functions, and the MON_LOCKWAIT administrative view instead of the SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function, the SNAPLOCK administrative view and SNAP_GET_LOCK table function, and the LOCKS_HELD administrative view which are deprecated in Fix Pack 1 of Version 9.7.

The schema is SYSPROC.

Table function parameters

search_args

An input parameter of type CLOB(1K) that represents a list of *key-value* pairs. If the list is empty or NULL, all locks in the currently connected database are returned. Otherwise, all locks that match all of the conditions represented by the list of *key-value* pairs are returned. A *key-value* pair must follow this format:

- A *key* is a string that consists of an opening tag, followed by the value, followed by a closing tag.
- An opening tag consists of an opening angle bracket, followed by the key name, followed by a closing angle bracket. No spaces are allowed.
- A closing tag consists of an opening angle bracket, followed by a forward slash, followed by the key name, followed by a closing angle bracket. No spaces are allowed.
- All keys are case-sensitive and can only be specified once in the *search_args* parameter.
- The order of the keys does not matter.

SQLCODE -171 is returned for an invalid *key-value* pair.

SQLCODE -204 is returned if the table does not exist.

An AND operation is performed between different keys. An OR operation is performed between multiple values of the same key. For example, the following use of the *search_args* parameter returns a list of all locks of type Table or Row, that are held, or waiting to be acquired, in either Shared or Exclusive mode, by the application with the handle 123:

```
CLOB('<application_handle>123</application_handle>
      <lock_object_type>Table:Row</lock_object_type>
      <lock_mode>S:X</lock_mode>')
```

The available keys for the MON_GET_LOCKS table function are as follows:

- application_handle

  Returns a list of all locks that are currently held or are in the process of being acquired by the specified application handle. Only a single occurrence of the key value can be specified. The value is specified as an INTEGER. For example:

  ```
  CLOB('<application_handle>145</application_handle>'))
  ```

- lock_name

  Returns a list of all locks that match the specified lock name. Only a single occurrence of the key value can be specified. The value is specified as a string of maximum length 32. For example:

  ```
  CLOB('<lock_name>00030005000000000280000452</lock_name>')
  ```

- lock_object_type

  Returns a list of all locks that match the specified lock object type. Multiple occurrences of the key value can be specified (to a maximum of 5). Each value (case insensitive) must be separated by a colon (:) and is specified as a string of a maximum length of 32 characters. For example:

  ```
  CLOB('<lock_object_type>Table:Chunk:Plan</lock_object_type>')
  ```

For a list of possible input values, see "lock_object_type - Lock object type waited on monitor element".

- lock_mode

  Returns a list of all locks that match the specified lock mode. Multiple occurrences of the key value can be specified (to a maximum of 5). Each value (case insensitive) is separated by a colon (:) and is specified as a string of maximum length 3. For example:

  ```
  CLOB('<lock_mode>IS:IN:U</lock_mode>')
  ```

  For a list of possible input values, see "lock_mode - Lock mode monitor element".

- lock_status

  Returns a list of all locks in the specified status. Only a single occurrence of the key value can be specified. The value is specified as a character.

  ```
  CLOB('<lock_status>W</lock_status>')
  ```

  For a list of possible input values, see "lock_status - Lock status monitor element".

- table_schema

  Returns a list of all locks that are qualified by the specified schema name. The table_name key must also be specified. Only a single occurrence of the key value can be specified. The value is specified as a string of maximum length 128.

- table_name

  Returns a list of all locks that reference the specified table. The table_schema key must also be specified. Only a single occurrence of the key value can be specified. The value is specified as a string of maximum length 128. For example:

  ```
  CLOB('<table_schema>USER1</table_schema>
        <table_name>INVENTORY</table_name>')
  ```

The following examples demonstrate how to use *key-value* pairs in the *search_args* parameter.

1. To search for all ROW and TABLE locks:

   ```
   CLOB('<lock_object_type>Table:Row</lock_object_type>')
   ```

2. To search for all locks that application handle 123 is holding or waiting to acquire that reference table T1, and were created by user USER1:

   ```
   CLOB('<application_handle>123</application_handle>
         <table_schema>USER1</table_schema>
         <table_name>T1</table_name>')
   ```

3. To search for all TABLE, ROW, and BUFFERPOOL locks that are currently held in Shared mode:

   ```
   CLOB('<lock_mode>S</lock_mode>
         <lock_status>G</lock_status>
         <lock_object_type>Table:Row:Reorg</lock_object_type>')
   ```

member
An input argument of type INTEGER that specifies from which member the data is returned. Specify -1 for the current member, and -2 for all active members.

Authorization

One of the following authorities or privilege is required:

- SYSADM authority

- SYSMON authority

Example

In this sample scenario, the MON_GET_LOCKS and MON_GET_APPL_LOCKWAIT table functions are used to investigate the locking situation in the current connected database, on all members.

1. Call the MON_GET_APPL_LOCKWAIT table function to determine all the locks that are waiting to be acquired in the current connected database, on all members:

```
SELECT lock_name,
       hld_member,
       lock_status,
       hld_application_handle FROM
       TABLE (MON_GET_APPL_LOCKWAIT(NULL, -2))
```

   This query returns the following output:

```
LOCK_NAME                  HLD_MEMBER LOCK_STATUS HLD_APPLICATION_HANDLE
-------------------------- ---------- ----------- ---------------------
0003000500000000280000452 -2          W
0003000500000000280000452 -2          W
0003000500000000280000452 -2          W

  3 record(s) selected.
```

   The records that show HLD_MEMBER is -2 indicate that the lock 0x0003000500000000280000452 is being held at a remote member.

2. Call the MON_GET_LOCKS table function to determine the holder of the lock, by specifying the lock name, 0x0003000500000000280000452, as the search argument:

```
SELECT lock_name,
       member,
       lock_status,
       application_handle FROM
    TABLE (MON_GET_LOCKS(
      CLOB('<lock_name>0003000500000000280000452</lock_name>),
        -2))
```

   This query returns the following output:

```
LOCK_NAME                  MEMBER LOCK_STATUS APPLICATION_HANDLE
-------------------------- ------ ----------- ------------------
0003000500000000280000452 0      W                12562
0003000500000000280000452 1      W                12562
0003000500000000280000452 2      G                65545
0003000500000000280000452 3      W                12562

  4 record(s) selected.
```

   To find out more about the application holding the lock, you can call the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 or WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 table functions.

Information returned

Table 127. Information returned by the MON_GET_LOCKS table function

Column name	Data type	Description or monitor element
APPLICATION_HANDLE	BIGINT	application_handle - Application handle If the LOCK_STATUS column is G, this represents the application that is currently holding the lock. If the LOCK_STATUS column is W or C, this represents the application that is currently waiting to acquire the lock.
MEMBER	SMALLINT	Database member from which the data was retrieved for this row.
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
LOCK_OBJECT_TYPE	VARCHAR(32)	lock_object_type - Lock object type If the LOCK_STATUS column is G, this represents the type of object that the application is currently holding. If the LOCK_STATUS column is W or C, then this represents the type of object that the application is currently waiting to acquire. For possible input values, see "lock_object_type - Lock object type waited on monitor element".
LOCK_OBJECT_TYPE_ID	CHAR(1) FOR BIT DATA	Reserved for internal use

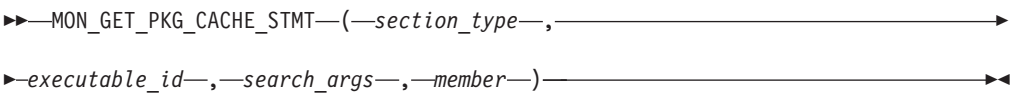
Table 127. Information returned by the MON_GET_LOCKS table function (continued)

Column name	Data type	Description or monitor element
LOCK_MODE	VARCHAR(3)	lock_mode - Lock mode If the LOCK_STATUS column is G, this represents the mode that the application is currently holding the lock in. If the LOCK_STATUS column is W or C, this represents the mode that the application is currently waiting to acquire the lock in. If the mode is unknown, a value of NULL is returned for this column.
LOCK_CURRENT_MODE	VARCHAR(3)	lock_current_mode - Original Lock Mode Before Conversion If the mode is unknown, a value of NULL is returned for this column.
LOCK_STATUS	CHAR(1)	lock_status - Lock status
LOCK_ATTRIBUTES	CHAR(16)	lock_attributes - Lock attributes
LOCK_RELEASE_FLAGS	CHAR(16)	Reserved for internal use
LOCK_RRIID	BIGINT	Reserved for internal use
LOCK_COUNT	BIGINT	Reserved for internal use
LOCK_HOLD_COUNT	BIGINT	Reserved for internal use
TBSP_ID	BIGINT	tablespace_id - Table space ID For locks that do not reference a table space, a value of NULL is returned.
TAB_FILE_ID	BIGINT	table_file_id - Table file ID
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for internal use

MON_GET_PKG_CACHE_STMT table function - Get SQL statement activity metrics in the package cache

The MON_GET_PKG_CACHE_STMT table function returns a point-in-time view of both static and dynamic SQL statements in the database package cache.

Syntax



The schema is SYSPROC.

Table function parameters

section_type
An optional input argument (either "D" or "S") of type CHAR(1) that specifies information type for the returned statement. If the argument is NULL or the empty string, information is returned for all SQL statements. Not case sensitive: “D” stands for dynamic; “S” for static.

executable_id
An optional input argument of type VARCHAR (32) for bit data that specifies a unique section of the database package cache. If a null value is specified, information is returned for all SQL statements. Note that when the *executable_id* is specified, the *section_type* argument is ignored. For example, if an *executable_id* is specified for a dynamic statement, the dynamic statement details will be returned by this table function even if *section_type* is specified as static ("S").

search_args
An optional input parameter of type CLOB(1K), that allows you to specify one or more optional search argument strings. For example:

```
'<modified_within>5</modified_within><update_boundary_time>myPkgEvmon
    </update_boundary_time>'
```

The available search argument tags are as follows:

- '<modified_within>X</modified_within>'

  Returns only those statement entries that have either been inserted into the cache or executed within the last *X* minutes (where *X* is a positive integer value). If the argument is not specified, all entries in the cache are returned.

- '<update_boundary_time>evmon_name</update_boundary_time>'

  Updates the event monitor boundary timestamp to the current time for the package cache event monitor specified by *evmon_name*. If this event monitor specifies `where updated_since_boundary_time` as an output criteria in its WHERE clause, only package cache entries that subsequently have their metrics updated are captured when evicted from the package cache. This operation only has an effect if the specified package cache event monitor is active when the command is issued.

Each input argument can be specified only once. The search argument tags must be specified in lowercase.

member
An optional input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set.

Authorization

EXECUTE privilege on the MON_GET_PKG_CACHE_STMT function.

Example

List all the dynamic SQL statements from the database package cache ordered by the average CPU time.

```
db2 SELECT MEMBER,
      SECTION_TYPE ,
      TOTAL_CPU_TIME/NUM_EXEC_WITH_METRICS as
      AVG_CPU_TIME,EXECUTABLE_ID
      FROM TABLE(MON_GET_PKG_CACHE_STMT ( 'D', NULL, NULL, -2)) as T
        WHERE T.NUM_EXEC_WITH_METRICS <> 0 ORDER BY AVG_CPU_TIME
```

The following is an example of output from this query.

```
MEMBER SECTION_TYPE AVG_CPU_TIME         EXECUTABLE_ID
------ ------------ -------------------- --------------------------------------------------------------------------
     0 D                             754 x'0100000000000000A0000000000000000000000020020081126171554951791'
     0 D                            2964 x'0100000000000000790000000000000000000000020020081126171533551120'
     0 D                            5664 x'0100000000000000C0000000000000000000000020020081126171720728997'
     0 D                            5723 x'0100000000000000B0000000000000000000000020020081126171657272914'
     0 D                            9762 x'0100000000000000D0000000000000000000000020020081126172409987719'

5 record(s) selected.
```

Note: It takes a longer time period to build the compilation environment and to transfer statement text (which can be as large as 2 MB) between members. To improve performance when retrieving a list of all the statements from the package cache, do not to select the STMT_TEXT and the COMP_ENV_DESC columns.

With the above output, we can use the *executable_id* to find out the details about the most expensive statement (in terms of the average CPU time):

```
db2 SELECT STMT_TEXT FROM TABLE(MON_GET_PKG_CACHE_STMT)
      (null, x'0100000000000000D0000000000000000000000020020081126172409987719', null, -2))

STMT_TEXT
--------------------------------------------------------------------------------------------
SELECT * FROM EMPLOYEE
```

As another example, assume a user named Alex has a connection associated to workload A which has the COLLECT ACTIVITY METRICS set. Another user, Brent, is associated to workload B that has the COLLECT ACTIVITY METRICS set to NONE. In addition, the database **mon_act_metrics** configuration parameter is set to NONE. When Brent executes the query:

```
SELECT count(*) FROM syscat.tables
```

all metrics are returned as 0 and the value of **num_exec_with_metrics** is also 0. Then Alex executes the same statement afterwards, but the metrics are collected this time for the execution of the statement and **num_exec_with_metrics** increments. So, after Brent and Alex execute that statement, the result of this query:

```
SELECT num_executions, num_exec_with_metrics, SUBSTR(stmt_text,1,50) AS stmt_text
    FROM TABLE (MON_GET_PKG_CACHE_STMT('d', null, null, -1)) AS tf
    WHERE stmt_text LIKE 'SELECT count%'
```

shows that the SELECT statement ran twice and one of the execution times had the activity metrics collected.

```
NUM_EXECUTIONS NUM_EXEC_WITH_METRICS STMT_TEXT
-------------- --------------------- --------------------
             2                     1 SELECT count(*) FROM syscat.tables

  1 record(s) selected.
```

Usage notes

The MON_GET_PKG_CACHE_STMT table function returns a point-in-time view of both static and dynamic SQL statements in the database package cache. This allows you to examine the aggregated metrics for a particular SQL statement, allowing you to quickly determine the reasons for poor query performance. The metrics returned are aggregates of the metrics gathered during each execution of the statement.

It also allows you to compare the behavior of an individual cached section, relative to the other statements, to assist in identifying the most expensive section or statements (in terms of the execution costs).

The activity metrics reported by this function are rolled up to the database cache at the end of the execution of the activity.

Metrics collection for the execution of any statement is controlled through the COLLECT ACTIVITY METRICS clause on workloads, or the **mon_act_metrics** database configuration parameter at the database level. Metrics are only collected for executions of the statement if the statement was submitted by a connection associated with a workload or database for which activity metrics are enabled. The **num_exec_with_metrics** element returned by the MON_GET_PKG_CACHE_STMT function indicates how many executions of the statement have had metrics collected and have contributed to the aggregate metrics reported. If no metrics are collected for any execution of the statement, then the **num_exec_with_metrics** element is 0 and all metric values are returned as 0.

Information returned

Table 128. Information returned for MON_GET_PKG_CACHE_STMT

Column Name	Data Type	Description or corresponding monitor element
MEMBER	SMALLINT	member - Database member
SECTION_TYPE	CHAR(1)	section_type - Section type indicator.
INSERT_TIMESTAMP	TIMESTAMP	insert_timestamp - Statement insert timestamp
EXECUTABLE_ID	VARCHAR(32) FOR BIT DATA	executable_id - Executable ID.
PACKAGE_NAME	VARCHAR(128)	package_name - Package name. This output is valid for static SQL statements only. A NULL value is returned if the statement is dynamic.
PACKAGE_SCHEMA	VARCHAR(128)	package_schema - Package schema. This output is valid for static SQL statements only. A NULL value is returned if the statement is dynamic.
PACKAGE_VERSION_ID	VARCHAR(64)	package_version_id - Package version. This output is valid for static SQL statements only. A NULL value is returned if the statement is dynamic or if you did not specify the package version for static statement. An empty string will be returned for static statement if the package version identifier was not specified by you when the package was created.
SECTION_NUMBER	BIGINT	section_number - Section number. A NULL value is returned if the statement is dynamic.

Table 128. Information returned for MON_GET_PKG_CACHE_STMT (continued)

Column Name	Data Type	Description or corresponding monitor element
EFFECTIVE_ISOLATION	CHAR(2)	effective_isolation - Effective isolation. This is the isolation value in effect for the section; it can be different from what it was originally requested at compilation time.
NUM_EXECUTIONS	BIGINT	num_executions - Statement executions
NUM_EXEC_WITH_METRICS	BIGINT	num_exec_with_metrics - Number of executions with metrics collected.
PREP_TIME	BIGINT	prep_time - Preparation time Note that PREP_TIME is only valid for dynamic SQL statements. PREP_TIME is reported as 0 for static SQL statements.
TOTAL_ACT_TIME	BIGINT	total_act_time - Total activity time
TOTAL_ACT_WAIT_TIME	BIGINT	total_act_wait_time - Total activity wait time
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
TOTAL_SECTION_SORT_TIME	BIGINT	total_section_sort_time - Total section sort time
TOTAL_SECTION_SORT_PROC_TIME	BIGINT	total_section_sort_proc_time - Total section sort processing time
TOTAL_SECTION_SORTS	BIGINT	total_section_sorts - Total section sorts
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
LOCK_WAITS	BIGINT	lock_waits - Lock waits
ROWS_MODIFIED	BIGINT	rows_modified - Rows modified
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_RETURNED	BIGINT	rows_returned - Rows returned
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads

Table 128. Information returned for MON_GET_PKG_CACHE_STMT (continued)

Column Name	Data Type	Description or corresponding monitor element
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer pool XDA data writes
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
TOTAL_SORTS	BIGINT	total_sorts - Total Sorts
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
WLM_QUEUE_TIME_TOTAL	BIGINT	wlm_queue_time_total - Workload manager total queue time
WLM_QUEUE_ASSIGNMENTS_TOTAL	BIGINT	wlm_queue_assignments_total - Workload manager total queue assignments
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
FCM_RECV_VOLUME	BIGINT	fcm_recv_volume - FCM recv volume
FCM_RECVS_TOTAL	BIGINT	fcm_recvs_total - FCM recvs total
FCM_SEND_VOLUME	BIGINT	fcm_send_volume - FCM send volume
FCM_SENDS_TOTAL	BIGINT	fcm_sends_total - FCM sends total
FCM_RECV_WAIT_TIME	BIGINT	fcm_recv_wait_time - FCM recv wait time
FCM_SEND_WAIT_TIME	BIGINT	fcm_send_wait_time - FCM send wait time
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
LOG_BUFFER_WAIT_TIME	BIGINT	log_buffer_wait_time - Log buffer wait time
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers
LOG_DISK_WAIT_TIME	BIGINT	log_disk_wait_time - Log disk wait time
LOG_DISK_WAITS_TOTAL	BIGINT	log_disk_waits_total - Log disk waits total
LAST_METRICS_UPDATE	TIMESTAMP	last_metrics_update - Metrics last update timestamp
NUM_COORD_EXEC	BIGINT	num_coord_exec - Number of executions by coordinator agent
NUM_COORD_EXEC_WITH_METRICS	BIGINT	num_coord_exec_with_metrics - Number of executions by coordinator agent
VALID	CHAR(1)	valid - Section validity indicator.
TOTAL_ROUTINE_TIME	BIGINT	total_routine_time - Total routine time
TOTAL_ROUTINE_INVOCATIONS	BIGINT	total_routine_invocations - Total routine invocations
ROUTINE_ID	BIGINT	Reserved for future use.

Table 128. Information returned for MON_GET_PKG_CACHE_STMT (continued)

Column Name	Data Type	Description or corresponding monitor element
STMT_TYPE_ID	VARCHAR(32)	stmt_type_id - Statement type identifier
QUERY_COST_ESTIMATE	BIGINT	query_cost_estimate - Query cost estimate
STMT_PKG_CACHE_ID	BIGINT	stmt_pkgcache_id - Statement package cache identifier
COORD_STMT_EXEC_TIME	BIGINT	coord_stmt_exec_time - Execution time for statement by coordinator agent
STMT_EXEC_TIME	BIGINT	stmt_exec_time - Statement execution time
TOTAL_SECTION_TIME	BIGINT	total_section_time - Total section time
TOTAL_SECTION_PROC_TIME	BIGINT	total_section_proc_time - Total section processing time
TOTAL_ROUTINE_NON _SECT_TIME	BIGINT	total_routine_non_sect_time - Non-section routine execution time
TOTAL_ROUTINE_NON _SECT_PROC_TIME	BIGINT	total_routine_non_sect_proc_time - Non-section processing time
STMT_TEXT	CLOB(2MB)	stmt_text - SQL statement text
COMP_ENV_DESC	BLOB(10K)	comp_env_desc - Compilation environment handle. You can use the existing COMPILATION_ENV table function to get the detailed compilation environment of the specific statement if needed.
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future additional metrics.

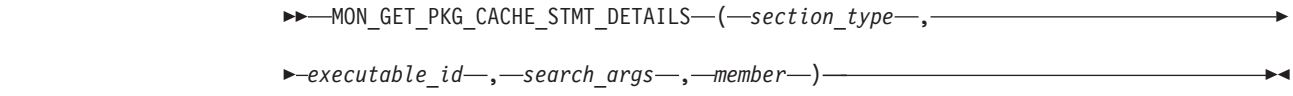
MON_GET_PKG_CACHE_STMT_DETAILS - get detailed metrics for package cache entries

The MON_GET_PKG_CACHE_STMT_DETAILS table function returns detailed metrics for one or more package cache entries.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

The metrics returned by the MON_GET_PKG_CACHE_STMT_DETAILS table function represent the accumulation of all metrics for statements in the package cache. Statement metrics are rolled up to the package cache upon activity completion.

Syntax



The schema is SYSPROC.

Table function parameters

section_type
An optional input argument (either "D" or "S") of type CHAR(1) that specifies information type for the returned statement. If the argument is NULL or an empty string, information is returned for all SQL statements. Not case sensitive: D stands for dynamic; S for static.

executable_id
An optional input argument of type VARCHAR (32) for bit data that specifies a unique section of the database package cache. If a null value is specified, information is returned for all SQL statements. When the *executable_id* is specified, the *section_type* argument is ignored. For example, if an *executable_id* is specified for a dynamic statement, the dynamic statement details will be returned by this table function even if *section_type* is specified as static ("S").

search_args
An optional input parameter of type CLOB(1K), that allows you to specify one or more optional search argument strings. For example:

```
'<modified_within>5</modified_within><update_boundary_time>myPkgEvmon
     </update_boundary_time>'
```

The available search argument tags are as follows:

- '<modified_within>X</modified_within>'

  Returns only those statement entries that have either been inserted into the cache or executed within the last *X* minutes (where *X* is a positive integer value). If the argument is not specified, all entries in the cache are returned.

- '<update_boundary_time>evmon_name</update_boundary_time>'

  Updates the event monitor boundary timestamp to the current time for the package cache event monitor specified by *evmon_name*. If this event monitor specifies `where updated_since_boundary_time` as an output criteria in its WHERE clause, only package cache entries that subsequently have their metrics updated are captured when evicted from the package cache. This operation only has an effect if the specified package cache event monitor is active when the command is issued.

- '<stmt_details>true</stmt_details>' or '<stmt_details>false</stmt_details>'

  Includes or excludes the *stmt_text* and *comp_env_desc* data in the resulting XML document. This allows you to exclude these relatively large portions of the document when you do not need them (for example, if you are using the XML document to provide input for the MON_FORMAT_XML_* table functions that return formatted row-based output). If this argument tag is not specified, the *stmt_text* and *comp_env_desc* data are included by default.

Each input argument can be specified only once. The search argument tags must be specified in lowercase.

member
An optional input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set.

Authorization

EXECUTE privilege on the MON_GET_PKG_CACHE_STMT_DETAILS function.

Examples

The first example demonstrates how to examine the package cache and select the 10 statements that have read and returned the largest number of rows. Additionally, the results show the cumulative amount of time spent executing each of these statements (in the STMT_EXEC_TIME output column).

```
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 40) STMT_TEXT,
       DETMETRICS.ROWS_RETURNED,
       DETMETRICS.STMT_EXEC_TIME
FROM TABLE(MON_GET_PKG_CACHE_STMT_DETAILS(CAST(NULL AS CHAR(1)),
    CAST(NULL AS VARCHAR(32) FOR BIT DATA),
    CAST(NULL AS CLOB(1K)), -1)) AS STMT_METRICS,
    XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
       '$DETMETRICS/db2_pkg_cache_stmt_details' PASSING
    XMLPARSE(DOCUMENT STMT_METRICS.DETAILS) as "DETMETRICS"
    COLUMNS "STMT_TEXT" CLOB PATH 'stmt_text',
        "ROWS_RETURNED" BIGINT PATH 'activity_metrics/rows_returned',
        "STMT_EXEC_TIME" BIGINT PATH 'activity_metrics/stmt_exec_time'
    ) AS DETMETRICS
ORDER BY rows_returned DESC
FETCH FIRST 10 ROWS ONLY
```

The following is an example of output from this query.

```
STMT_TEXT                              ROWS_RETURNED STMT_EXEC_TIME
-------------------------------------- ------------- --------------
SELECT CREATOR, NAME, CTIME FROM SYSIBM.           134             38
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 4            44            336
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 4            10            333
SELECT COLNAME, TYPENAME FROM  SYSCAT.CO            10              6
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 4            10            334
SELECT TRIGNAME FROM  SYSCAT.TRIGGERS WH             8              1
SELECT COUNT(*) FROM SYSCAT.TABLESPACES              2              0
SELECT POLICY FROM SYSTOOLS.POLICY WHERE             1              0
CALL SYSPROC.POLICY_INSTALL ('I','DB2Tab             1             62
CALL SYSPROC.POLICY_INSTALL ('I','DB2Tab             1             64

  10 record(s) selected.
```

The second example shows, for dynamic SQL statements that have waited on a lock while executing, the number of executions, number of lock waits and average time spent per lock wait. The output shows values accumulated over the lifetime of the package cache entries, but restricts information to statements that have executed within the last minute (by setting the modified_within argument tag to 1). The query excludes the statement details (*stmt_text* and *comp_env_desc* data) because they are not required and they are computationally expensive to report (by setting the stmt_details argument tag to false).

```
SELECT NUM_EXEC_WITH_METRICS, LOCK_WAITS,
  (LOCK_WAIT_TIME / LOCK_WAITS) AVG_LOCK_WAIT_TIME
FROM TABLE(MON_GET_PKG_CACHE_STMT_DETAILS('D', CAST(NULL
   AS VARCHAR(32) FOR BIT DATA),
   CLOB(
      '<modified_within>1</modified_within><stmt_details>false</stmt_details>')
       , -1))
   AS STMT_METRICS,
   XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
      '$DETMETRICS/db2_pkg_cache_stmt_details' PASSING
   XMLPARSE(DOCUMENT STMT_METRICS.DETAILS) as "DETMETRICS"
   COLUMNS "NUM_EXEC_WITH_METRICS" BIGINT PATH 'num_exec_with_metrics',
      "LOCK_WAITS" BIGINT PATH 'lock_waits',
      "LOCK_WAIT_TIME" BIGINT PATH 'activity_metrics/lock_wait_time'
   ) AS DETMETRICS
WHERE LOCK_WAITS <> 0
ORDER BY AVG_LOCK_WAIT_TIME DESC
```

The following is an example of output from this query.

```
NUM_EXEC_WITH_METRICS LOCK_WAITS           AVG_LOCK_WAIT_TIME
--------------------- -------------------- --------------------
                    4                    2                  139
                    9                    3                   90
```

Usage notes

The metrics returned by this function represent the accumulation of all metrics for statements in the package cache. Statement metrics are rolled up to the package cache upon activity completion.

Information returned

Table 129. Information returned for MON_GET_PKG_CACHE_STMT_DETAILS

Column Name	Data Type	Description or corresponding monitor element
MEMBER	SMALLINT	member - Database member
SECTION_TYPE	CHAR(1)	section_type - Section type indicator.
EXECUTABLE_ID	VARCHAR(32) FOR BIT DATA	executable_id - Executable ID.
DETAILS	BLOB(8M)	XML document containing detailed metrics for the unit of work. See Table 130 for a description of the elements in this document.

Table 130. Detailed metrics returned for MON_GET_PKG_CACHE_STMT_DETAILS

Element Name	Data Type	Description
member	xs:short	member - Database member
valid	xs:string(1)	valid - Section validity indicator
executable_id	xs:hexBinary(32)	executable_id - Executable ID.
section_type	xs:string(1)	section_type - Section type indicator.
num_executions	xs:nonNegativeInteger	num_executions - Statement executions
num_exec_with_metrics	xs:nonNegativeInteger	num_exec_with_metrics - Number of executions with metrics collected.
prep_time	xs:nonNegativeInteger	prep_time - Preparation time Note that PREP_TIME is only valid for dynamic SQL statements. PREP_TIME is reported as 0 for static SQL statements.
effective_isolation	xs:string(2)	effective_isolation - Effective isolation. This is the isolation value in effect for the section; it can be different from what it was originally requested at compilation time.
stmt_pkgcache_id	xs:long	stmt_pkgcache_id - Statement package cache identifier
query_cost_estimate	xs:long	query_cost_estimate - Query cost estimate
stmt_type_id	xs:string	stmt_type_id - Statement type identifier
insert_timestamp	xs:dateTime	insert_timestamp - Statement insert timestamp
last_metrics_update	xs:dateTime	last_metrics_update - Metrics last update timestamp
package_name	xs:string(128)	package_name - Package name. This output is valid for static SQL statements only. A NULL value is returned if the statement is dynamic.

Table 130. Detailed metrics returned for MON_GET_PKG_CACHE_STMT_DETAILS (continued)

Element Name	Data Type	Description
package_schema	xs:string(128)	package_schema - Package schema. This output is valid for static SQL statements only. A NULL value is returned if the statement is dynamic.
package_version_id	xs:string(64)	package_version_id - Package version. This output is valid for static SQL statements only. This element is not produced if the statement is dynamic or if you did not specify the package version for static statement. If you did not specify the package version identifier when the package was created, an empty string is returned for a static statement.
section_number	xs:short	section_number - Section number. This element is not produced if the statement is dynamic.
stmt_text	xs:string(2097152)	stmt_text - SQL statement text
comp_env_desc	xs:hexBinary(10240)	comp_env_desc - Compilation environment handle. You can use the existing COMPILATION_ENV table function to get the detailed compilation environment of the specific statement if needed.
wlm_queue_time_total	xs:long	wlm_queue_time_total - Workload manager total queue time
wlm_queue_assignments_total	xs:long	wlm_queue_assignments_total - Workload manager total queue assignments
fcm_tq_recv_wait_time	xs:long	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
fcm_message_recv_wait_time	xs:long	fcm_message_recv_wait_time - FCM message recv wait time
fcm_tq_send_wait_time	xs:long	fcm_tq_send_wait_time - FCM tablequeue send wait time
fcm_message_send_wait_time	xs:long	fcm_message_send_wait_time - FCM message send wait time
lock_wait_time	xs:long	lock_wait_time - Time waited on locks
lock_waits	xs:long	lock_waits - Lock waits
direct_read_time	xs:long	direct_read_time - Direct read time
direct_read_reqs	xs:long	direct_read_reqs - Direct read requests
direct_write_time	xs:long	direct_write_time - Direct write time
direct_write_reqs	xs:long	direct_write_reqs - Direct write requests
log_buffer_wait_time	xs:long	log_buffer_wait_time - Log buffer wait time
num_log_buffer_full	xs:long	num_log_buffer_full - Number of full log buffers
log_disk_wait_time	xs:long	log_disk_wait_time - Log disk wait time
log_disk_waits_total	xs:long	log_disk_waits_total - Log disk waits total
pool_write_time	xs:long	pool_write_time - Total buffer pool physical write time
pool_read_time	xs:long	pool_read_time - Total buffer pool physical read time
audit_file_write_wait_time	xs:long	audit_file_write_wait_time - Audit file write wait time
audit_file_writes_total	xs:long	audit_file_writes_total - Total Audit files written
audit_subsystem_wait_time	xs:long	audit_subsystem_wait_time - Audit subsystem wait time

Table 130. Detailed metrics returned for MON_GET_PKG_CACHE_STMT_DETAILS (continued)

Element Name	Data Type	Description
audit_subsystem_waits_total	xs:long	audit_subsystem_waits_total - Total audit subsystem waits
diaglog_write_wait_time	xs:long	diaglog_write_wait_time - Diag log write time
diaglog_writes_total	xs:long	diaglog_writes_total - Diag log total writes
fcm_send_wait_time	xs:long	fcm_send_wait_time - FCM send wait time
fcm_recv_wait_time	xs:long	fcm_recv_wait_time - FCM recv wait time
total_act_wait_time	xs:long	total_act_wait_time - Total activity wait time
total_section_sort_proc_time	xs:long	total_section_sort_proc_time - Total section sort processing time
total_section_sort_time	xs:long	total_section_sort_time - Total section sort time
total_section_sorts	xs:long	total_section_sorts - Total section sorts
total_act_time	xs:long	total_act_time - Total activity time
rows_read	xs:long	rows_read - Rows read
rows_modified	xs:long	rows_modified - Rows modified
pool_data_l_reads	xs:long	pool_data_l_reads - Buffer pool data logical reads
pool_index_l_reads	xs:long	pool_index_l_reads - Buffer pool index logical reads
pool_temp_data_l_reads	xs:long	pool_temp_data_l_reads - Buffer pool temporary data logical reads
pool_temp_index_l_reads	xs:long	pool_temp_index_l_reads - Buffer pool temporary index logical reads
total_cpu_time	xs:long	total_cpu_time - Total CPU time
pool_data_p_reads	xs:long	pool_data_p_reads - Buffer pool data physical reads
pool_temp_data_p_reads	xs:long	pool_temp_data_p_reads - Buffer pool temporary data physical reads
pool_xda_p_reads	xs:long	pool_xda_p_reads - Buffer pool XDA data physical reads
pool_temp_xda_p_reads	xs:long	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
pool_index_p_reads	xs:long	pool_index_p_reads - Buffer pool index physical reads
pool_temp_index_p_reads	xs:long	pool_temp_index_p_reads - Buffer pool temporary index physical reads
pool_data_writes	xs:long	pool_data_writes - Buffer pool data writes
pool_xda_writes	xs:long	pool_xda_writes - Buffer pool XDA data writes
pool_index_writes	xs:long	pool_index_writes - Buffer pool index writes
direct_reads	xs:long	direct_reads - Direct reads from database
direct_writes	xs:long	direct_writes - Direct writes to database
rows_returned	xs:long	rows_returned - Rows returned
deadlocks	xs:long	deadlocks - Deadlocks detected
lock_timeouts	xs:long	lock_timeouts - Number of lock timeouts
lock_escals	xs:long	lock_escals - Number of lock escalations
fcm_sends_total	xs:long	fcm_sends_total - FCM sends total
fcm_recvs_total	xs:long	fcm_recvs_total - FCM recvs total

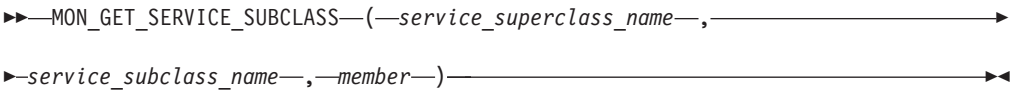
Table 130. Detailed metrics returned for MON_GET_PKG_CACHE_STMT_DETAILS (continued)

Element Name	Data Type	Description
fcm_send_volume	xs:long	fcm_send_volume - FCM send volume
fcm_recv_volume	xs:long	fcm_recv_volume - FCM recv volume
fcm_message_sends_total	xs:long	fcm_message_sends_total - FCM message sends total
fcm_message_recvs_total	xs:long	fcm_message_recvs_total - FCM message recvs total
fcm_message_send_volume	xs:long	fcm_message_send_volume - FCM message send volume
fcm_message_recv_volume	xs:long	fcm_message_recv_volume - FCM message recv volume
fcm_tq_sends_total	xs:long	fcm_tq_sends_total - FCM tablequeue send total
fcm_tq_recvs_total	xs:long	fcm_tq_recvs_total - FCM tablequeue recvs total
fcm_tq_send_volume	xs:long	fcm_tq_send_volume - FCM tablequeue send volume
fcm_tq_recv_volume	xs:long	fcm_tq_recv_volume - FCM tablequeue recv volume
tq_tot_send_spills	xs:long	tq_tot_send_spills - Total number of table queue buffers overflowed
post_threshold_sorts	xs:long	post_threshold_sorts - Post threshold sorts
post_shrthreshold_sorts	xs:long	post_shrthreshold_sorts - Post shared threshold sorts
sort_overflows	xs:long	sort_overflows - Sort overflows
audit_events_total	xs:long	audit_events_total - Total audit events
total_sorts	xs:long	total_sorts - Total Sorts
stmt_exec_time	xs:long	stmt_exec_time - Statement execution time
coord_stmt_exec_time	xs:long	coord_stmt_exec_time - Execution time for statement by coordinator agent
total_routine_non_ sect_proc_time	xs:long	total_routine_non_sect_proc_time - Non-section processing time
total_routine_non_sect_time	xs:long	total_routine_non_sect_time - Non-section routine execution time
total_section_proc_time	xs:long	total_section_proc_time - Total section processing time
total_section_time	xs:long	total_section_time - Total section time
total_app_section_executions	xs:long	total_app_section_executions - Total section executions
total_routine_user_ code_proc_time	xs:long	total_routine_user_code_proc_time - Total routine user code processing time
total_routine_user_code_time	xs:long	total_routine_user_code_time - Total routine user code time
total_routine_time	xs:long	total_routine_time - Total routine time
num_coord_exec	xs:long	num_coord_exec - Number of executions by coordinator agent
num_coord_exec_with_metrics	xs:long	num_coord_exec_with_metrics - Number of executions by coordinator agent with metrics
num_thresh_violations	xs:long	num_threshold_violations - Number of threshold violations
num_lw_thresh_exceeded	xs:long	num_lw_thresh_exceeded - Number of thresholds exceeded
total_routine_invocations	xs:long	total_routine_invocations - Total routine invocations

MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics

The MON_GET_SERVICE_SUBCLASS table function returns metrics for one or more service subclasses.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies a valid service superclass name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all the superclasses in the database.

service_subclass_name
An input argument of type VARCHAR(128) that specifies a valid service subclass name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all the subclasses in the database.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_SERVICE_SUBCLASS function.

Example

Display the total CPU time used and total number of requests processed for each service class, ordered by CPU usage.

```
SELECT varchar(service_superclass_name,30) as service_superclass,
                  varchar(service_subclass_name,30) as service_subclass,
                  sum(total_cpu_time) as total_cpu,
                  sum(app_rqsts_completed_total) as total_rqsts
FROM TABLE(MON_GET_SERVICE_SUBCLASS('','',-2)) AS t
GROUP BY service_superclass_name, service_subclass_name
ORDER BY total_cpu desc
```

The following is an example of output from this query.

```
SERVICE_SUPERCLASS             SERVICE_SUBCLASS               ...
------------------------------ ------------------------------ ...
SYSDEFAULTUSERCLASS            SYSDEFAULTSUBCLASS             ...
SYSDEFAULTMAINTENANCECLASS     SYSDEFAULTSUBCLASS             ...
SYSDEFAULTSYSTEMCLASS          SYSDEFAULTSUBCLASS             ...

  3 record(s) selected.
```

Output for query (continued).

```
... TOTAL_CPU            TOTAL_RQSTS
... -------------------- --------------------
...               967673                  100
.. .                   0                    0
...                    0                    0
```

Usage notes

The metrics returned by the MON_GET_SERVICE_SUBCLASS table function represent the accumulation of all metrics for requests that have executed under the indicated service subclass. Metrics are rolled up to a service class on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_SERVICE_SUBCLASS table function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses and the *mon_req_metrics* database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default, request metrics are enabled at the database level. If request metrics are disabled at the database level and for a service superclass, the metrics reported for each connection mapped to that service superclass stop increasing (or remain at 0 if request metrics were disabled at database activation time).

The MON_GET_SERVICE_SUBCLASS table function returns one row of data per service subclass and per member. No aggregation across service classes (on a member), or across members (for a service class or more), is performed. However, aggregation can be achieved through SQL queries as shown in the example. The input parameters have the effect of being ANDed together. Therefore, if you specify conflicting input parameters (for example, a superclass name SUPA and subclass name SUBB that is not a subclass of SUPA), no rows are returned.

Tip: A request might execute in more than one service subclass. For example, this situation might occur if a request is mapped from one service subclass to another by using a Workload Manager (WLM) threshold with a REMAP ACTIVITY action. Although the time spent metrics are updated for each service subclass under which the request executes, the request counters are incremented for the service subclass where the request completed. Therefore, you should not analyze the averages of request times for a single subclass. All subclasses to which an activity can be mapped must be analyzed in conjunction with one another. For example, if a threshold exists that can map activities from service subclass A to service subclass B, then when you compute averages of requests, you should aggregate the counters and metrics for service subclasses A and B, and compute the averages using the aggregates.

Information returned

Table 131. Information returned for MON_GET_SERVICE_SUBCLASS

Column Name	Data Type	Description or corresponding monitor element
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	service_superclass_name - Service superclass name
SERVICE_SUBCLASS_NAME	VARCHAR(128)	service_subclass_name - Service subclass name
SERVICE_CLASS_ID	INTEGER	service_class_id - Service class ID
MEMBER	SMALLINT	member - Database member
ACT_ABORTED_TOTAL	BIGINT	act_aborted_total - Total aborted activities
ACT_COMPLETED_TOTAL	BIGINT	act_completed_total - Total completed activities
ACT_REJECTED_TOTAL	BIGINT	act_rejected_total - Total rejected activities
AGENT_WAIT_TIME	BIGINT	agent_wait_time - Agent wait time
AGENT_WAITS_TOTAL	BIGINT	agent_waits_total - Total agent waits
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer pool XDA data writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time

Table 131. Information returned for MON_GET_SERVICE_SUBCLASS (continued)

Column Name	Data Type	Description or corresponding monitor element
CLIENT_IDLE_WAIT_TIME	BIGINT	client_idle_wait_time - Client idle wait time
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
FCM_RECV_VOLUME	BIGINT	fcm_recv_volume - FCM recv volume
FCM_RECVS_TOTAL	BIGINT	fcm_recvs_total - FCM recvs total
FCM_SEND_VOLUME	BIGINT	fcm_send_volume - FCM send volume
FCM_SENDS_TOTAL	BIGINT	fcm_sends_total - FCM sends total
FCM_RECV_WAIT_TIME	BIGINT	fcm_recv_wait_time - FCM recv wait time
FCM_SEND_WAIT_TIME	BIGINT	fcm_send_wait_time - FCM send wait time
IPC_RECV_VOLUME	BIGINT	ipc_recv_volume - Interprocess communication recv volume
IPC_RECV_WAIT_TIME	BIGINT	ipc_recv_wait_time - Interprocess communication recv wait time
IPC_RECVS_TOTAL	BIGINT	ipc_recvs_total - Interprocess communication recvs total
IPC_SEND_VOLUME	BIGINT	ipc_send_volume - Interprocess communication send volume
IPC_SEND_WAIT_TIME	BIGINT	ipc_send_wait_time - Interprocess communication send wait time
IPC_SENDS_TOTAL	BIGINT	ipc_sends_total - Interprocess communication send total
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOG_BUFFER_WAIT_TIME	BIGINT	log_buffer_wait_time - Log buffer wait time
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers
LOG_DISK_WAIT_TIME	BIGINT	log_disk_wait_time - Log disk wait time
LOG_DISK_WAITS_TOTAL	BIGINT	log_disk_waits_total - Log disk waits total
RQSTS_COMPLETED_TOTAL	BIGINT	rqsts_completed_total - Total requests completed
ROWS_MODIFIED	BIGINT	rows_modified - Rows modified
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_RETURNED	BIGINT	rows_returned - Rows returned
TCPIP_RECV_VOLUME	BIGINT	tcpip_recv_volume - TCP/IP received volume
TCPIP_SEND_VOLUME	BIGINT	tcpip_send_volume - TCP/IP send volume

Table 131. Information returned for MON_GET_SERVICE_SUBCLASS (continued)

Column Name	Data Type	Description or corresponding monitor element
TCPIP_RECV_WAIT_TIME	BIGINT	tcpip_recv_wait_time - TCP/IP recv wait time
TCPIP_RECVS_TOTAL	BIGINT	tcpip_recvs_total - TCP/IP recvs total
TCPIP_SEND_WAIT_TIME	BIGINT	tcpip_send_wait_time - TCP/IP send wait time
TCPIP_SENDS_TOTAL	BIGINT	tcpip_sends_total - TCP/IP sends total
TOTAL_APP_RQST_TIME	BIGINT	total_app_rqst_time - Total application request time
TOTAL_RQST_TIME	BIGINT	total_rqst_time - Total request time
WLM_QUEUE_TIME_TOTAL	BIGINT	wlm_queue_time_total - Workload manager total queue time
WLM_QUEUE_ASSIGNMENTS _TOTAL	BIGINT	wlm_queue_assignments_total - Workload manager total queue assignments
TOTAL_RQST_MAPPED_IN	BIGINT	total_rqst_mapped_in - Total request mapped-in
TOTAL_RQST_MAPPED_OUT	BIGINT	total_rqst_mapped_out - Total request mapped-out
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
TOTAL_WAIT_TIME	BIGINT	total_wait_time - Total wait time
APP_RQSTS_COMPLETED_TOTAL	BIGINT	app_rqsts_completed_total - Total application requests completed
TOTAL_SECTION_SORT_TIME	BIGINT	total_section_sort_time - Total section sort time
TOTAL_SECTION_SORT_PROC_TIME	BIGINT	total_section_sort_proc_time - Total section sort processing time
TOTAL_SECTION_SORTS	BIGINT	total_section_sorts - Total section sorts
TOTAL_SORTS	BIGINT	total_sorts - Total Sorts
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_COMPILE_TIME	BIGINT	total_compile_time - Total compile time
TOTAL_COMPILE_PROC_TIME	BIGINT	total_compile_proc_time - Total compile processing time
TOTAL_COMPILATIONS	BIGINT	total_compilations - Total compilations
TOTAL_IMPLICIT_COMPILE_TIME	BIGINT	total_implicit_compile_time - Total implicit compile time
TOTAL_IMPLICIT_COMPILE _PROC_TIME	BIGINT	total_implicit_compile_proc_time - Total implicit compile processing time
TOTAL_IMPLICIT_COMPILATIONS	BIGINT	total_implicit_compilations - Total implicit complications
TOTAL_SECTION_TIME	BIGINT	total_section_time - Total section time
TOTAL_SECTION_PROC_TIME	BIGINT	total_section_proc_time - Total section processing time

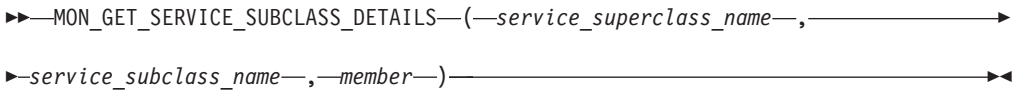
Table 131. Information returned for MON_GET_SERVICE_SUBCLASS (continued)

Column Name	Data Type	Description or corresponding monitor element
TOTAL_APP_SECTION_EXECUTIONS	BIGINT	total_app_section_executions - Total section executions
TOTAL_ACT_TIME	BIGINT	total_act_time - Total activity time
TOTAL_ACT_WAIT_TIME	BIGINT	total_act_wait_time - Total activity wait time
ACT_RQSTS_TOTAL	BIGINT	act_rqsts_total - Total activity requests
TOTAL_ROUTINE_TIME	BIGINT	total_routine_time - Total routine time
TOTAL_ROUTINE_INVOCATIONS	BIGINT	total_routine_invocations - Total routine invocations
TOTAL_COMMIT_TIME	BIGINT	total_commit_time - Total commit time
TOTAL_COMMIT_PROC_TIME	BIGINT	total_commit_proc_time - Total commits processing time
TOTAL_APP_COMMITS	BIGINT	total_app_commits - Total application commits
INT_COMMITS	BIGINT	int_commits - Internal commits
TOTAL_ROLLBACK_TIME	BIGINT	total_rollback_time - Total rollback time
TOTAL_ROLLBACK_PROC_TIME	BIGINT	total_rollback_proc_time - Total rollback processing time
TOTAL_APP_ROLLBACKS	BIGINT	total_app_rollbacks - Total application rollbacks
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
TOTAL_RUNSTATS_TIME	BIGINT	total_runstats_time - Total runtime statistics
TOTAL_RUNSTATS_PROC_TIME	BIGINT	total_runstats_proc_time - Total runtime statistics processing time
TOTAL_RUNSTATS	BIGINT	total_runstats - Total runtime statistics
TOTAL_REORG_TIME	BIGINT	total_reorg_time - Total reorganization time
TOTAL_REORG_PROC_TIME	BIGINT	total_reorg_proc_time - Total reorganization processing time
TOTAL_REORGS	BIGINT	total_reorgs - Total reorganizations
TOTAL_LOAD_TIME	BIGINT	total_load_time - Total load time
TOTAL_LOAD_PROC_TIME	BIGINT	total_load_proc_time - Total load processing time
TOTAL_LOADS	BIGINT	total_loads - Total loads
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
THRESH_VIOLATIONS	BIGINT	thresh_violations - Number of threshold violations
NUM_LW_THRESH_EXCEEDED	BIGINT	num_lw_thresh_exceeded - Number of thresholds exceeded
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use

MON_GET_SERVICE_SUBCLASS_DETAILS table function - Get detailed service subclass metrics

The MON_GET_SERVICE_SUBCLASS_DETAILS table function returns detailed metrics for one or more service subclasses.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies a valid service superclass name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all the superclasses in the database.

service_subclass_name
An input argument of type VARCHAR(128) that specifies a valid service subclass name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all the subclasses in the database.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_SERVICE_SUBCLASS_DETAILS function.

Example

Display the total CPU time used and total number of requests processed for each service superclass, ordered by CPU usage in relational format (using XMLTABLE).

```
SELECT varchar(scmetrics.service_superclass_name,30) as service_superclass,
       sum(detmetrics.total_cpu_time) as total_cpu,
       sum(detmetrics.app_rqsts_completed_total) as total_rqsts
FROM TABLE(MON_GET_SERVICE_SUBCLASS_DETAILS('','',-2)) AS SCMETRICS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
        '$detmetric/db2_service_subclass'
        PASSING XMLPARSE(DOCUMENT SCMETRICS.DETAILS)
        as "detmetric"
COLUMNS "TOTAL_CPU_TIME" INTEGER PATH 'system_metrics/total_cpu_time',
        "APP_RQSTS_COMPLETED_TOTAL" INTEGER PATH
        'system_metrics/app_rqsts_completed_total')
        AS DETMETRICS
GROUP BY service_superclass_name
ORDER BY total_cpu desc
```

The following is an example of output from this query.

```
SERVICE_SUPERCLASS             TOTAL_CPU            TOTAL_RQSTS
------------------------------ -------------------- --------------------
SYSDEFAULTUSERCLASS                         2428188                   26
SYSDEFAULTMAINTENANCECLASS                        0                    0
SYSDEFAULTSYSTEMCLASS                             0                    0

  3 record(s) selected.
```

Usage notes

The metrics returned by the MON_GET_SERVICE_SUBCLASS_DETAILS table function represent the accumulation of all metrics for requests that have executed under the indicated service subclass. This function is similar to the MON_GET_SERVICE_SUBCLASS table function:

- The the MON_GET_SERVICE_SUBCLASS table function returns the most commonly used metrics in a column based format and is the most performance efficient method of retrieving metrics.
- The MON_GET_SERVICE_SUBCLASS_DETAILS table function returns the entire set of available metrics in an XML document format, which provides maximum flexibility for formatting output. The XML-based output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function (see the example).

Metrics are rolled up to a service class on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_SERVICE_SUBCLASS_DETAILS table function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses, and the **mon_req_metrics** database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default request metrics are enabled at the database level. If request metrics are disabled at the database level and for a service superclass, the metrics reported for each connection mapped to that service superclass stop increasing (or remain at 0 if request metrics were disabled at database activation time).

The MON_GET_SERVICE_SUBCLASS_DETAILS table function returns one row of data per service subclass and per member. No aggregation across service classes (on a member), or across members (for a service class or more), is performed. However, aggregation can be achieved through SQL queries (see the example). The input parameters have the effect of being ANDed together. Therefore, if you specify conflicting input parameters (for example, a superclass name SUPA and subclass name SUBB that is not a subclass of SUPA), no rows are returned.

Tip: A request might execute in more than one service subclass. For example, this situation might occur if a request is mapped from one service subclass to another by using a Workload Manager (WLM) threshold with a REMAP ACTIVITY action. Although the time spent metrics are updated for each service subclass under which the request executes, the request counters are incremented for the service subclass where the request completed. Therefore, you should not analyze the averages of request times for a single subclass. All subclasses to which an activity can be

mapped must be analyzed in conjunction with one another. For example, if a threshold exists that can map activities from service subclass A to service subclass B, then when you compute averages of requests, you should aggregate the counters and metrics for service subclasses A and B, and compute the averages using the aggregates.

The schema for the XML document that is returned in the DETAILS column is available in the file `sqllib/misc/DB2MonRoutines.xsd`. Further details can be found in the file `sqllib/misc/DB2MonCommon.xsd`.

Information returned

Table 132. Information returned for MON_GET_SERVICE_SUBCLASS_DETAILS

Column Name	Data Type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	service_superclass_name - Service superclass name
SERVICE_SUBCLASS_NAME	VARCHAR(128)	service_subclass_name - Service subclass name
SERVICE_CLASS_ID	INTEGER	service_class_id - Service class ID
MEMBER	SMALLINT	member - Database member
DETAILS	BLOB(1M)	XML document that contains detailed metrics for the service class. See Table 133 for a description of the elements in this document.

The following example shows the structure of the XML document that is returned in the DETAILS column.

```
<db2_service_subclass xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="90700000">
   <service_superclass_name>SYSDEFAULTSYSTEMCLASS</service_superclass_name>
   <service_subclass_name>SYSDEFAULTSUBCLASS</service_subclass_name>
   <service_subclass_id>11</service_subclass_id>
   <member>0</member>
   <system_metrics release="90700000">
   <act_aborted_total>5</act_aborted_total>
    ...
   <wlm_queue_assignments_total>3</wlm_queue_assignments_total>
   </system_metrics>
</db2_service_subclass>
```

For the full schema, see `sqllib/misc/DB2MonRoutines.xsd`.

Table 133. Detailed metrics returned for MON_GET_SERVICE_SUBCLASS_DETAILS

Element Name	Data Type	Description or corresponding monitor element
service_superclass_name	xs:string(128)	service_superclass_name - Service superclass name
service_subclass_name	xs:string(128)	service_subclass_name - Service subclass name
service_class_id	xs:nonNegativeInteger	service_class_id - Service class ID
member	xs:nonNegativeInteger	member - Database member
act_aborted_total	xs:nonNegativeInteger	act_aborted_total - Total aborted activities
act_completed_total	xs:nonNegativeInteger	act_completed_total - Total completed activities
act_rejected_total	xs:nonNegativeInteger	act_rejected_total - Total rejected activities
act_rqsts_total	xs:nonNegativeInteger	act_rqsts_total - Total activity requests
agent_wait_time	xs:nonNegativeInteger	agent_wait_time - Agent wait time
agent_waits_total	xs:nonNegativeInteger	agent_waits_total - Total agent waits

Table 133. Detailed metrics returned for MON_GET_SERVICE_SUBCLASS_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
app_rqsts_completed_total	xs:nonNegativeInteger	app_rqsts_completed_total - Total application requests completed
audit_events_total	xs:nonNegativeInteger	audit_events_total - Total audit events
audit_subsystem_wait_time	xs:nonNegativeInteger	audit_subsystem_wait_time - Audit subsystem wait time
audit_subsystem_waits_total	xs:nonNegativeInteger	audit_subsystem_waits_total - Total audit subsystem waits
audit_file_write_wait_time	xs:nonNegativeInteger	audit_file_write_wait_time - Audit file write wait time
audit_file_writes_total	xs:nonNegativeInteger	audit_file_writes_total - Total Audit files written
cat_cache_inserts	xs:nonNegativeInteger	cat_cache_inserts - Catalog cache inserts
cat_cache_lookups	xs:nonNegativeInteger	cat_cache_lookups - Catalog cache lookups
client_idle_wait_time	xs:nonNegativeInteger	client_idle_wait_time - Client idle wait time
deadlocks	xs:nonNegativeInteger	deadlocks - Deadlocks detected
diaglog_writes_total	xs:nonNegativeInteger	diaglog_writes_total - Diag log total writes
diaglog_write_wait_time	xs:nonNegativeInteger	diaglog_write_wait_time - Diag log write time
direct_read_time	xs:nonNegativeInteger	direct_read_time - Direct read time
direct_write_time	xs:nonNegativeInteger	direct_write_time - Direct write time
direct_read_reqs	xs:nonNegativeInteger	direct_read_reqs - Direct read requests
direct_reads	xs:nonNegativeInteger	direct_reads - Direct reads from database
direct_write_reqs	xs:nonNegativeInteger	direct_write_reqs - Direct write requests
direct_writes	xs:nonNegativeInteger	direct_writes - Direct writes to database
fcm_recv_volume	xs:nonNegativeInteger	fcm_recv_volume - FCM recv volume
fcm_recv_wait_time	xs:nonNegativeInteger	fcm_recv_wait_time - FCM recv wait time
fcm_recvs_total	xs:nonNegativeInteger	fcm_recvs_total - FCM recvs total
fcm_message_recv_volume	xs:nonNegativeInteger	fcm_message_recv_volume - FCM message recv volume
fcm_message_recvs_total	xs:nonNegativeInteger	fcm_message_recvs_total - FCM message recvs total
fcm_message_recv_wait_time	xs:nonNegativeInteger	fcm_message_recv_wait_time - FCM message recv wait time
fcm_message_send_volume	xs:nonNegativeInteger	fcm_message_send_volume - FCM message send volume
fcm_message_send_wait_time	xs:nonNegativeInteger	fcm_message_send_wait_time - FCM message send wait time
fcm_message_sends_total	xs:nonNegativeInteger	fcm_message_sends_total - FCM message sends total
fcm_send_volume	xs:nonNegativeInteger	fcm_send_volume - FCM send volume
fcm_send_wait_time	xs:nonNegativeInteger	fcm_send_wait_time - FCM send wait time
fcm_sends_total	xs:nonNegativeInteger	fcm_sends_total - FCM sends total
fcm_tq_recv_wait_time	xs:nonNegativeInteger	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
fcm_tq_send_wait_time	xs:nonNegativeInteger	fcm_tq_send_wait_time - FCM tablequeue send wait time
fcm_tq_recv_volume	xs:nonNegativeInteger	fcm_tq_recv_volume - FCM tablequeue recv volume
fcm_tq_recvs_total	xs:nonNegativeInteger	fcm_tq_recvs_total - FCM tablequeue recvs total
fcm_tq_send_volume	xs:nonNegativeInteger	fcm_tq_send_volume - FCM tablequeue send volume
fcm_tq_sends_total	xs:nonNegativeInteger	fcm_tq_sends_total - FCM tablequeue send total
int_commits	xs:nonNegativeInteger	int_commits - Internal commits
int_rollbacks	xs:nonNegativeInteger	int_rollbacks - Internal rollbacks

Table 133. Detailed metrics returned for MON_GET_SERVICE_SUBCLASS_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of tablequeue buffers overflowed
ipc_recv_volume	xs:nonNegativeInteger	ipc_recv_volume - Interprocess communication recv volume
ipc_recv_wait_time	xs:nonNegativeInteger	ipc_recv_wait_time - Interprocess communication recv wait time
ipc_recvs_total	xs:nonNegativeInteger	ipc_recvs_total - Interprocess communication recvs total
ipc_send_volume	xs:nonNegativeInteger	ipc_send_volume - Interprocess communication send volume
ipc_send_wait_time	xs:nonNegativeInteger	ipc_send_wait_time - Interprocess communication send wait time
ipc_sends_total	xs:nonNegativeInteger	ipc_sends_total - Interprocess communication send total
lock_escals	xs:nonNegativeInteger	lock_escals - Number of lock escalations
lock_timeouts	xs:nonNegativeInteger	lock_timeouts - Number of lock timeouts
lock_wait_time	xs:nonNegativeInteger	lock_wait_time - Time waited on locks
lock_waits	xs:nonNegativeInteger	lock_waits - Lock waits
log_buffer_wait_time	xs:nonNegativeInteger	log_buffer_wait_time - Log buffer wait time
log_disk_wait_time	xs:nonNegativeInteger	log_disk_wait_time - Log disk wait time
log_disk_waits_total	xs:nonNegativeInteger	log_disk_waits_total - Log disk waits total
num_lw_thresh_exceeded	xs:nonNegativeInteger	num_lw_thresh_exceeded - Number of thresholds exceeded
pkg_cache_inserts	xs:nonNegativeInteger	pkg_cache_inserts - Package cache inserts
pkg_cache_lookups	xs:nonNegativeInteger	pkg_cache_lookups - Package cache lookups
pool_data_l_reads	xs:nonNegativeInteger	pool_data_l_reads - Buffer pool data logical reads
pool_data_p_reads	xs:nonNegativeInteger	pool_data_p_reads - Buffer pool data physical reads
pool_data_writes	xs:nonNegativeInteger	pool_data_writes - Buffer pool data writes
pool_index_l_reads	xs:nonNegativeInteger	pool_index_l_reads - Buffer pool index logical reads
pool_index_p_reads	xs:nonNegativeInteger	pool_index_p_reads - Buffer pool index physical reads
pool_index_writes	xs:nonNegativeInteger	pool_index_writes - Buffer pool index writes
pool_read_time	xs:nonNegativeInteger	pool_read_time - Total buffer pool physical read time
pool_temp_data_l_reads	xs:nonNegativeInteger	pool_temp_data_l_reads - Buffer pool temporary data logical reads
pool_temp_data_p_reads	xs:nonNegativeInteger	pool_temp_data_p_reads - Buffer pool temporary data physical reads
pool_temp_index_l_reads	xs:nonNegativeInteger	pool_temp_index_l_reads - Buffer pool temporary index logical reads
pool_temp_index_p_reads	xs:nonNegativeInteger	pool_temp_index_p_reads - Buffer pool temporary index physical reads
pool_temp_xda_l_reads	xs:nonNegativeInteger	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
pool_temp_xda_p_reads	xs:nonNegativeInteger	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
pool_write_time	xs:nonNegativeInteger	pool_write_time - Total buffer pool physical write time

Table 133. Detailed metrics returned for MON_GET_SERVICE_SUBCLASS_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
pool_xda_l_reads	xs:nonNegativeInteger	pool_xda_l_reads - Buffer pool XDA data logical reads
pool_xda_p_reads	xs:nonNegativeInteger	pool_xda_p_reads - Buffer pool XDA data physical reads
pool_xda_writes	xs:nonNegativeInteger	pool_xda_writes - Buffer pool XDA data writes
num_log_buffer_full	xs:nonNegativeInteger	num_log_buffer_full - Number of full log buffers
rqsts_completed_total	xs:nonNegativeInteger	rqsts_completed_total - Total requests completed
total_rqst_mapped_in	xs:nonNegativeInteger	total_rqst_mapped_in - Total request mapped-in
total_rqst_mapped_out	xs:nonNegativeInteger	total_rqst_mapped_out - Total request mapped-out
rows_modified	xs:nonNegativeInteger	rows_modified - Rows modified
rows_read	xs:nonNegativeInteger	rows_read - Rows read
rows_returned	xs:nonNegativeInteger	rows_returned - Rows returned
tcpip_recv_volume	xs:nonNegativeInteger	tcpip_recv_volume - TCP/IP received volume
tcpip_recv_wait_time	xs:nonNegativeInteger	tcpip_recv_wait_time - TCP/IP recv wait time
tcpip_recvs_total	xs:nonNegativeInteger	tcpip_recvs_total - TCP/IP recvs total
tcpip_send_volume	xs:nonNegativeInteger	tcpip_send_volume - TCP/IP send volume
tcpip_send_wait_time	xs:nonNegativeInteger	tcpip_send_wait_time - TCP/IP send wait time
tcpip_sends_total	xs:nonNegativeInteger	tcpip_sends_total - TCP/IP sends total
thresh_violations	xs:nonNegativeInteger	thresh_violations - Number of threshold violations
total_act_time	xs:nonNegativeInteger	total_act_time - Total activity time
total_act_wait_time	xs:nonNegativeInteger	total_act_wait_time - Total activity wait time
total_app_commits	xs:nonNegativeInteger	total_app_commits - Total application commits
total_app_rollbacks	xs:nonNegativeInteger	total_app_rollbacks - Total application rollbacks
total_app_rqst_time	xs:nonNegativeInteger	total_app_rqst_time - Total application request time
total_app_section_executions	xs:nonNegativeInteger	total_app_section_executions - Total section executions
total_commit_proc_time	xs:nonNegativeInteger	total_commit_proc_time - Total commits processing time
total_commit_time	xs:nonNegativeInteger	total_commit_time - Total commit time
total_compilations	xs:nonNegativeInteger	total_compilations - Total compilations
total_compile_proc_time	xs:nonNegativeInteger	total_compile_proc_time - Total compile processing time
total_compile_time	xs:nonNegativeInteger	total_compile_time - Total compile time
total_cpu_time	xs:nonNegativeInteger	total_cpu_time - Total CPU time
total_implicit_compilations	xs:nonNegativeInteger	total_implicit_compilations - Total implicit complications
total_implicit_compile _proc_time	xs:nonNegativeInteger	total_implicit_compile_proc_time - Total implicit compile processing time
total_implicit_compile_time	xs:nonNegativeInteger	total_implicit_compile_time - Total implicit compile time
total_loads	xs:nonNegativeInteger	total_loads - Total loads
total_load_proc_time	xs:nonNegativeInteger	total_load_proc_time - Total load processing time
total_load_time	xs:nonNegativeInteger	total_load_time - Total load time
total_reorgs	xs:nonNegativeInteger	total_reorgs - Total reorganizations
total_reorg_proc_time	xs:nonNegativeInteger	total_reorg_proc_time - Total reorganization processing time
total_reorg_time	xs:nonNegativeInteger	total_reorg_time - Total reorganization time

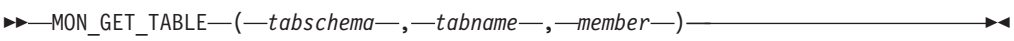
Table 133. Detailed metrics returned for MON_GET_SERVICE_SUBCLASS_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
total_rollback_proc_time	xs:nonNegativeInteger	total_rollback_proc_time - Total rollback processing time
total_rollback_time	xs:nonNegativeInteger	total_rollback_time - Total rollback time
total_routine_invocations	xs:nonNegativeInteger	total_routine_invocations - Total routine invocations
total_routine_time	xs:nonNegativeInteger	total_routine_time - Total routine time
total_routine_user_ code_proc_time	xs:nonNegativeInteger	total_routine_user_code_proc_time - Total routine user code processing time
total_routine_user_ code_time	xs:nonNegativeInteger	total_routine_user_code_time - Total routine user code time
total_rqst_time	xs:nonNegativeInteger	total_rqst_time - Total request time
total_runstats	xs:nonNegativeInteger	total_runstats - Total runtime statistics
total_runstats_proc_time	xs:nonNegativeInteger	total_runstats_proc_time - Total runtime statistics processing time
total_runstats_time	xs:nonNegativeInteger	total_runstats_time - Total runtime statistics
total_section_proc_time	xs:nonNegativeInteger	total_section_proc_time - Total section processing time
total_section_sort_time	xs:nonNegativeInteger	total_section_sort_time - Total section sort time
total_section_sort_proc_time	xs:nonNegativeInteger	total_section_sort_proc_time - Total section sort processing time
total_section_sorts	xs:nonNegativeInteger	total_section_sorts - Total section sorts
total_section_time	xs:nonNegativeInteger	total_section_time - Total section time
total_sorts	xs:nonNegativeInteger	total_sorts - Total Sorts
post_threshold_sorts	xs:nonNegativeInteger	post_shrthreshold_sorts - Post shared threshold sorts
post_shrthreshold_sorts	xs:nonNegativeInteger	post_shrthreshold_sorts - Post shared threshold sorts
sort_overflows	xs:nonNegativeInteger	sort_overflows - Sort overflows
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of table queue buffers overflowed
total_wait_time	xs:nonNegativeInteger	total_wait_time - Total wait time
wlm_queue_time_total	xs:nonNegativeInteger	wlm_queue_time_total - Workload manager total queue time
wlm_queue_assignments_total	xs:nonNegativeInteger	wlm_queue_assignments_total - Workload manager total queue assignments

MON_GET_TABLE table function - get table metrics

The MON_GET_TABLE table function returns monitor metrics for one or more tables.

Syntax



The schema is SYSPROC.

Table function parameters

tabschema
An input argument of type VARCHAR(128) that specifies a valid table schema name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all tables in all schemas in the database. If the argument is specified, metrics are only returned for tables in the specified schema.

tabname
An input argument of type VARCHAR(128) that specifies a valid table name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all the tables in the database.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the NULL value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_TABLE function.

Example

List the activity on all tables accessed since the database was activated, aggregated across all database members, ordered by highest number of reads.

```
SELECT varchar(tabschema,20) as tabschema,
       varchar(tabname,20) as tabname,
       sum(rows_read) as total_rows_read,
       sum(rows_inserted) as total_rows_inserted,
       sum(rows_updated) as total_rows_updated,
       sum(rows_deleted) as total_rows_deleted
FROM TABLE(MON_GET_TABLE('','',-2)) AS t
GROUP BY tabschema, tabname
ORDER BY total_rows_read DESC
```

The following is an example of output from this query.

```
TABSCHEMA            TABNAME              TOTAL_ROWS_READ      ...
-------------------- -------------------- -------------------- ...
SYSIBM               SYSHISTO                             113 ...
SYSIBM               SYSWORKL                              22 ...
SYSIBM               SYSROUTI                              13 ...
SYSIBM               SYSSERVI                              13 ...
SYSIBM               SYSTHRES                               6 ...
SYSIBM               SYSTABLE                               3 ...
SYSIBM               SYSCONTE                               2 ...
SYSIBM               SYSDBAUT                               2 ...
SYSIBM               SYSEVENT                               2 ...
SYSIBM               SYSPLAN                                1 ...
SYSIBM               SYSSURRO                               1 ...
SYSIBM               SYSVERSI                               1 ...
SYSIBM               SYSXMLST                               1 ...
SYSIBM               SYSAUDIT                               0 ...
SYSIBM               SYSROLEA                               0 ...
SYSIBM               SYSROLES                               0 ...
SYSIBM               SYSTASKS                               0 ...
SYSIBM               SYSWORKA                               0 ...
SYSIBM               SYSXMLPA                               0 ...

  19 record(s) selected.
```

Output for query (continued).

```
... TOTAL_ROWS_INSERTED  TOTAL_ROWS_UPDATED   TOTAL_ROWS_DELETED
... -------------------- -------------------- --------------------
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
```

Usage notes

The MON_GET_TABLE table function returns one row of data per database table and per database member. If range-partitioned tables are being used, one row is returned for each table partition per database member. No aggregation across database members is performed. However, aggregation can be achieved through SQL queries as shown in the example.

Metrics are returned only for tables accessed since the database was activated.

Metrics are always enabled. You do not need to turn on any system monitor switches to access table metrics through this function.

Information returned

Table 134. Information returned for MON_GET_TABLE

Column Name	Data Type	Description
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
MEMBER	SMALLINT	member - Database member
TAB_TYPE	VARCHAR(14)	table_type - Table type. This interface returns a text identifier based on defines in sqlmon.h, and is one of: • USER_TABLE • DROPPED_TABLE • TEMP_TABLE • CATALOG_TABLE • REORG_TABLE
TAB_FILE_ID	BIGINT	table_file_id - Table file ID
DATA_PARTITION_ID	INTEGER	data_partition_id - Data partition identifier
TBSP_ID	BIGINT	tablespace_id - Table space identification
INDEX_TBSP_ID	BIGINT	index_tbsp_id - Index table space ID
LONG_TBSP_ID	BIGINT	long_tbsp_id - Long table space ID

Table 134. Information returned for MON_GET_TABLE (continued)

Column Name	Data Type	Description
TABLE_SCANS	BIGINT	table_scans - Table scans
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
OVERFLOW_ACCESSES	BIGINT	overflow_accesses - Accesses to overflowed records
OVERFLOW_CREATES	BIGINT	overflow_creates - Overflow creates
PAGE_REORGS	BIGINT	page_reorgs - Page reorganizations
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

MON_GET_TABLESPACE table function - Get table space metrics

The MON_GET_TABLESPACE table function returns monitor metrics for one or more table spaces.

Syntax

The schema is SYSPROC.

Table function parameters

tbsp_name
An input argument of type VARCHAR(128) that specifies a valid table space name in the currently connected database when calling this function. If the argument is null or an empty string, metrics are retrieved for all table spaces in the database.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the NULL value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_TABLESPACE function.

Example

List table spaces ordered by number of physical reads from table space containers.

```
SELECT varchar(tbsp_name, 30) as tbsp_name,
                 member,
                 tbsp_type,
                 pool_data_p_reads
FROM TABLE(MON_GET_TABLESPACE('',-2)) AS t
ORDER BY pool_data_p_reads DESC
```

The following is an example of output from this query.

```
TBSP_NAME                      MEMBER TBSP_TYPE  POOL_DATA_P_READS
------------------------------ ------ ---------- --------------------
SYSCATSPACE                         0 DMS                          79
USERSPACE1                          0 DMS                          34
TEMPSPACE1                          0 SMS                           0

  3 record(s) selected.
```

Usage notes

The MON_GET_TABLESPACE table function returns one row of data per database table space and per database member. No aggregation across database members is performed. However, aggregation can be achieved through SQL queries.

Metrics collected by this function are controlled at the database level using the mon_obj_metrics configuration parameter. By default, metrics collection is enabled.

Information returned

Table 135. Information returned for MON_GET_TABLESPACE

Column Name	Data Type	Description or corresponding monitor element
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
MEMBER	SMALLINT	member - Database member
TBSP_TYPE	VARCHAR(10)	tablespace_type - Table space type. This interface returns a text identifier based on defines in sqlutil.h, and is one of: • DMS • SMS
TBSP_CONTENT_TYPE	VARCHAR(10)	tablespace_content_type - Table space content type. This interface returns a text identifier based on defines in sqlmon.h, and is one of: • ANY • LARGE • SYSTEMP • USRTEMP
TBSP_PAGE_SIZE	BIGINT	tablespace_page_size - Table space page size
TBSP_EXTENT_SIZE	BIGINT	tablespace_extent_size - Table space extent size
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size
TBSP_CUR_POOL_ID	BIGINT	tablespace_cur_pool_id - Buffer pool currently being used
TBSP_NEXT_POOL_ID	BIGINT	tablespace_next_pool_id - Buffer pool that will be used at next startup
FS_CACHING	SMALLINT	fs_caching - File system caching
TBSP_REBALANCER_MODE	VARCHAR(30)	tablespace_rebalancer_mode - Rebalancer mode. This interface returns a text identifier based on defines in sqlmon.h, and is one of: • NO_REBAL • FWD_REBAL • REV_REBAL • FWD_REBAL_OF_2PASS • REV_REBAL_OF_2PASS

Table 135. Information returned for MON_GET_TABLESPACE (continued)

Column Name	Data Type	Description or corresponding monitor element
TBSP_USING_AUTO_STORAGE	SMALLINT	tablespace_using_auto_storage - Table space enabled for automatic storage
TBSP_AUTO_RESIZE_ENABLED	SMALLINT	tablespace_auto_resize_enabled - Table space automatic resizing enabled
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer pool XDA data logical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer pool XDA data writes
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_ASYNC_DATA_READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes

Table 135. Information returned for MON_GET_TABLESPACE (continued)

Column Name	Data Type	Description or corresponding monitor element
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_ASYNC_INDEX_READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer pool asynchronous XDA data reads
POOL_ASYNC_XDA_READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer pool asynchronous XDA read requests
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer pool asynchronous XDA data writes
VECTORED_IOS	BIGINT	vectored_ios - Number of vectored IO requests
PAGES_FROM_VECTORED_IOS	BIGINT	pages_from_vectored_ios - Total number of pages read by vectored IO
BLOCK_IOS	BIGINT	block_ios - Number of block IO requests
PAGES_FROM_BLOCK_IOS	BIGINT	pages_from_block_ios - Total number of pages read by block IO
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
FILES_CLOSED	BIGINT	files_closed - Database files closed
TBSP_STATE	VARCHAR(256)	tablespace_state - Table space state
TBSP_USED_PAGES	BIGINT	tablespace_used_pages - Used pages in table space
TBSP_FREE_PAGES	BIGINT	tablespace_free_pages - Free pages in table space
TBSP_USABLE_PAGES	BIGINT	tablespace_usable_pages - Usable pages in table space
TBSP_TOTAL_PAGES	BIGINT	tablespace_total_pages - Total pages in table space
TBSP_PENDING_FREE_PAGES	BIGINT	tablespace_pending_free_pages - Pending free pages in table space
TBSP_PAGE_TOP	BIGINT	tablespace_page_top - Table space high watermark
TBSP_MAX_PAGE_TOP	BIGINT	tbsp_max_page_top - Maximum table space page high watermark
RECLAIMABLE_SPACE_ENABLED	SMALLINT	reclaimable_space_enabled - Reclaimable space enabled indicator
AUTO_STORAGE_HYBRID	SMALLINT	auto_storage_hybrid - Hybrid automatic storage table space indicator
TBSP_PATHS_DROPPED	SMALLINT	tablespace_paths_dropped - Table space using dropped path
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

MON_GET_UNIT_OF_WORK table function - Get unit of work metrics

The MON_GET_UNIT_OF_WORK table function returns metrics for one or more units of work.

Syntax

The schema is SYSPROC.

Table function parameters

application_handle
An optional input argument of type BIGINT that specifies a valid application handle in the same database as the one currently connected to when calling this function. If the argument is null, metrics are retrieved for units of work running in all superclasses in the database.

member
An optional input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the NULL value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_UNIT_OF_WORK function.

Example

Identify the units of work that are consuming the highest amount of CPU time on the system.

```
SELECT application_handle,
       uow_id,
       total_cpu_time,
       app_rqsts_completed_total,
       rqsts_completed_total
FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,-1)) AS t
ORDER BY total_cpu_time DESC
```

The following is an example of output from this query.

```
APPLICATION_HANDLE   UOW_ID      TOTAL_CPU_TIME       ...
-------------------- ----------- -------------------- ...
                  46           5                27959 ...

  1 record(s) selected.
```

Output for query (continued).

```
... APP_RQSTS_COMPLETED_TOTAL RQSTS_COMPLETED_TOTAL
... ------------------------- --------------------
...                        72                   48
```

Usage notes

The metrics returned by the MON_GET_UNIT_OF_WORK table function represent the accumulation of all metrics for requests that were submitted during a unit of work. Metrics are rolled up periodically during the unit of work. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an in interval of time, use the function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses and the *mon_req_metrics* database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default, request metrics are enabled at the database level. If request metrics have been disabled at the database level, and for a service superclass, the metrics reported for each unit of work mapped to that service superclass stop increasing (or remain at 0 if request metrics were disabled at database activation time).

The MON_GET_UNIT_OF_WORK table function returns one row of data per unit of work and per member. No aggregation across units of work (on a member), or across members (for a service class or more), is performed. However, aggregation can be achieved through SQL queries. The input parameters have the effect of being ANDed together.

Information returned

Table 136. Information returned for MON_GET_UNIT_OF_WORK

Column Name	Data Type	Description or corresponding monitor element
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	service_superclass_name - Service superclass name
SERVICE_SUBCLASS_NAME	VARCHAR(128)	service_subclass_name - Service subclass name
SERVICE_CLASS_ID	INTEGER	service_class_id - Service class ID
MEMBER	SMALLINT	member - Database member
COORD_MEMBER	SMALLINT	coord_member - Coordinator member
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
APPLICATION_ID	VARCHAR(128)	appl_id - Application ID
WORKLOAD_NAME	VARCHAR(128)	workload_name - Workload name
WORKLOAD_OCCURRENCE_ID	INTEGER	workload_occurrence_id - Workload occurrence identifier. This ID does not uniquely identify the workload occurrence unless it is coupled with the coordinator member and the workload name.
UOW_ID	INTEGER	uow_id - Unit of work ID
WORKLOAD_OCCURRENCE_STATE	VARCHAR(32)	workload_occurrence_state - Workload occurrence state
CLIENT_WRKSTNNAME	VARCHAR(255)	CURRENT CLIENT_WRKSTNNAME special register
CLIENT_ACCTNG	VARCHAR(255)	CURRENT CLIENT_ACCTNG special register
CLIENT_USERID	VARCHAR(255)	CURRENT CLIENT_USERID special register
CLIENT_APPLNAME	VARCHAR(255)	CURRENT CLIENT_APPLNAME special register
UOW_START_TIME	TIMESTAMP	uow_start_time - Unit of Work Start Timestamp
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
ACT_ABORTED_TOTAL	BIGINT	act_aborted_total - Total aborted activities
ACT_COMPLETED_TOTAL	BIGINT	act_completed_total - Total completed activities
ACT_REJECTED_TOTAL	BIGINT	act_rejected_total - Total rejected activities
AGENT_WAIT_TIME	BIGINT	agent_wait_time - Agent wait time
AGENT_WAITS_TOTAL	BIGINT	agent_waits_total - Total agent waits

Table 136. Information returned for MON_GET_UNIT_OF_WORK (continued)

Column Name	Data Type	Description or corresponding monitor element
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer pool XDA data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer pool XDA data writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
CLIENT_IDLE_WAIT_TIME	BIGINT	client_idle_wait_time - Client idle wait time
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
FCM_RECV_VOLUME	BIGINT	fcm_recv_volume - FCM recv volume
FCM_RECVS_TOTAL	BIGINT	fcm_recvs_total - FCM recvs total
FCM_SEND_VOLUME	BIGINT	fcm_send_volume - FCM send volume
FCM_SENDS_TOTAL	BIGINT	fcm_sends_total - FCM sends total
FCM_RECV_WAIT_TIME	BIGINT	fcm_recv_wait_time - FCM recv wait time
FCM_SEND_WAIT_TIME	BIGINT	fcm_send_wait_time - FCM send wait time

Table 136. Information returned for MON_GET_UNIT_OF_WORK (continued)

Column Name	Data Type	Description or corresponding monitor element
IPC_RECV_VOLUME	BIGINT	ipc_recv_volume - Interprocess communication recv volume
IPC_RECV_WAIT_TIME	BIGINT	ipc_recv_wait_time - Interprocess communication recv wait time
IPC_RECVS_TOTAL	BIGINT	ipc_recvs_total - Interprocess communication recvs total
IPC_SEND_VOLUME	BIGINT	ipc_send_volume - Interprocess communication send volume
IPC_SEND_WAIT_TIME	BIGINT	ipc_send_wait_time - Interprocess communication send wait time
IPC_SENDS_TOTAL	BIGINT	ipc_sends_total - Interprocess communication send total
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOG_BUFFER_WAIT_TIME	BIGINT	log_buffer_wait_time - Log buffer wait time
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers
LOG_DISK_WAIT_TIME	BIGINT	log_disk_wait_time - Log disk wait time
LOG_DISK_WAITS_TOTAL	BIGINT	log_disk_waits_total - Log disk waits total
NUM_LOCKS_HELD	BIGINT	locks_held - Locks held
RQSTS_COMPLETED_TOTAL	BIGINT	rqsts_completed_total - Total requests completed
ROWS_MODIFIED	BIGINT	rows_modified - Rows modified
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_RETURNED	BIGINT	rows_returned - Rows returned
TCPIP_RECV_VOLUME	BIGINT	tcpip_recv_volume - TCP/IP received volume
TCPIP_SEND_VOLUME	BIGINT	tcpip_send_volume - TCP/IP send volume
TCPIP_RECV_WAIT_TIME	BIGINT	tcpip_recv_wait_time - TCP/IP recv wait time
TCPIP_RECVS_TOTAL	BIGINT	tcpip_recvs_total - TCP/IP recvs total
TCPIP_SEND_WAIT_TIME	BIGINT	tcpip_send_wait_time - TCP/IP send wait time
TCPIP_SENDS_TOTAL	BIGINT	tcpip_sends_total - TCP/IP sends total
TOTAL_APP_RQST_TIME	BIGINT	total_app_rqst_time - Total application request time
TOTAL_RQST_TIME	BIGINT	total_rqst_time - Total request time
WLM_QUEUE_TIME_TOTAL	BIGINT	wlm_queue_time_total - Workload manager total queue time
WLM_QUEUE_ASSIGNMENTS _TOTAL	BIGINT	wlm_queue_assignments_total - Workload manager total queue assignments
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
TOTAL_WAIT_TIME	BIGINT	total_wait_time - Total wait time
APP_RQSTS_COMPLETED_TOTAL	BIGINT	app_rqsts_completed_total - Total application requests completed
TOTAL_SECTION_SORT_TIME	BIGINT	total_section_sort_time - Total section sort time

Table 136. Information returned for MON_GET_UNIT_OF_WORK (continued)

Column Name	Data Type	Description or corresponding monitor element
TOTAL_SECTION_SORT_PROC_TIME	BIGINT	total_section_sort_proc_time - Total section sort processing time
TOTAL_SECTION_SORTS	BIGINT	total_section_sorts - Total section sorts
TOTAL_SORTS	BIGINT	total_sorts - Total Sorts
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_COMPILE_TIME	BIGINT	total_compile_time - Total compile time
TOTAL_COMPILE_PROC_TIME	BIGINT	total_compile_proc_time - Total compile processing time
TOTAL_COMPILATIONS	BIGINT	total_compilations - Total compilations
TOTAL_IMPLICIT_COMPILE_TIME	BIGINT	total_implicit_compile_time - Total implicit compile time
TOTAL_IMPLICIT_COMPILE _PROC_TIME	BIGINT	total_implicit_compile_proc_time - Total implicit compile processing time
TOTAL_IMPLICIT_COMPILATIONS	BIGINT	total_implicit_compilations - Total implicit complications
TOTAL_SECTION_TIME	BIGINT	total_section_time - Total section time
TOTAL_SECTION_PROC_TIME	BIGINT	total_section_proc_time - Total section processing time
TOTAL_APP_SECTION_EXECUTIONS	BIGINT	total_app_section_executions - Total section executions
TOTAL_ACT_TIME	BIGINT	total_act_time - Total activity time
TOTAL_ACT_WAIT_TIME	BIGINT	total_act_wait_time - Total activity wait time
ACT_RQSTS_TOTAL	BIGINT	act_rqsts_total - Total activity requests
TOTAL_ROUTINE_TIME	BIGINT	total_routine_time - Total routine time
TOTAL_ROUTINE_INVOCATIONS	BIGINT	total_routine_invocations - Total routine invocations
TOTAL_COMMIT_TIME	BIGINT	total_commit_time - Total commit time
TOTAL_COMMIT_PROC_TIME	BIGINT	total_commit_proc_time - Total commits processing time
TOTAL_APP_COMMITS	BIGINT	total_app_commits - Total application commits
INT_COMMITS	BIGINT	int_commits - Internal commits
TOTAL_ROLLBACK_TIME	BIGINT	total_rollback_time - Total rollback time
TOTAL_ROLLBACK_PROC_TIME	BIGINT	total_rollback_proc_time - Total rollback processing time
TOTAL_APP_ROLLBACKS	BIGINT	total_app_rollbacks - Total application rollbacks
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
TOTAL_RUNSTATS_TIME	BIGINT	total_runstats_time - Total runtime statistics
TOTAL_RUNSTATS_PROC_TIME	BIGINT	total_runstats_proc_time - Total runtime statistics processing time
TOTAL_RUNSTATS	BIGINT	total_runstats - Total runtime statistics

Table 136. Information returned for MON_GET_UNIT_OF_WORK (continued)

Column Name	Data Type	Description or corresponding monitor element
TOTAL_REORG_TIME	BIGINT	total_reorg_time - Total reorganization time
TOTAL_REORG_PROC_TIME	BIGINT	total_reorg_proc_time - Total reorganization processing time
TOTAL_REORGS	BIGINT	total_reorgs - Total reorganizations
TOTAL_LOAD_TIME	BIGINT	total_load_time - Total load time
TOTAL_LOAD_PROC_TIME	BIGINT	total_load_proc_time - Total load processing time
TOTAL_LOADS	BIGINT	total_loads - Total loads
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
THRESH_VIOLATIONS	BIGINT	thresh_violations - Number of threshold violations
NUM_LW_THRESH_EXCEEDED	BIGINT	num_lw_thresh_exceeded - Number of thresholds exceeded
UOW_LOG_SPACE_USED	BIGINT	uow_log_space_used - Unit of Work Log Space Used
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics

The MON_GET_UNIT_OF_WORK_DETAILS table function returns detailed metrics for one or more units of work.

Syntax

The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies a valid application handle in the same database as the one currently connected to when calling this function. If the argument is null, metrics are retrieved for units of work running in all superclasses in the database.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the NULL value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_UNIT_OF_WORK_DETAILS function.

Example

Identify the units of work that are consuming the highest amount of CPU time on the system.

```
SELECT detmetrics.application_handle,
    detmetrics.uow_id,
    detmetrics.total_cpu_time,
    detmetrics.app_rqsts_completed_total,
    detmetrics.rqsts_completed_total
FROM TABLE(MON_GET_UNIT_OF_WORK_DETAILS(NULL,-2)) AS UOWMETRICS,
XMLTABLE (
    XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
        '$detmetric/db2_unit_of_work' PASSING
        XMLPARSE(DOCUMENT UOWMETRICS.DETAILS)
        as "detmetric"
COLUMNS
    "APPLICATION_HANDLE" INTEGER PATH 'application_handle',
    "UOW_ID" INTEGER PATH 'uow_id',
    "TOTAL_CPU_TIME" INTEGER PATH 'system_metrics/total_cpu_time',
    "APP_RQSTS_COMPLETED_TOTAL" INTEGER
        PATH 'system_metrics/app_rqsts_completed_total',
    "RQSTS_COMPLETED_TOTAL" INTEGER
        PATH 'system_metrics/rqsts_completed_total'
) AS DETMETRICS
ORDER BY total_cpu_time DESC
```

The following is an example of output from this query.

```
APPLICATION_HANDLE   UOW_ID      TOTAL_CPU_TIME       ...
-------------------- ----------- -------------------- ...
                  46           5                27959 ...

  1 record(s) selected.
```

Output for query (continued).

```
... APP_RQSTS_COMPLETED_TOTAL RQSTS_COMPLETED_TOTAL
... ------------------------- --------------------
...                        72                   48
```

Usage notes

The metrics returned by the MON_GET_UNIT_OF_WORK_DETAILS function represent the accumulation of all metrics for requests that were submitted during a unit of work. This function is similar to the MON_GET_UNIT_OF_WORK table function:

- The MON_GET_UNIT_OF_WORK table function returns the most commonly used metrics in a column based format and is the most performance efficient method of retrieving metrics.
- The MON_GET_UNIT_OF_WORK_DETAILS table function returns the entire set of available metrics in an XML document format, which provides maximum flexibility for formatting output. The XML based output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function (see the example).

Metrics are rolled up periodically during the unit of work. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_UNIT OF WORK_DETAILS table function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses and the **mon_req_metrics** database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default request metrics are enabled at the database level. If request metrics have been disabled at the database level, and for a service superclass, the metrics reported for each unit of work that are mapped to that service superclass stop increasing (or remain at 0 if request metrics were disabled at database activation time).

The MON_GET_UNIT_OF_WORK_DETAILS table function returns one row of data per unit of work and per member. No aggregation across units of work (on a member), or across members (for a service class or more), is performed. However, aggregation can be achieved through SQL queries. The input parameters have the effect of being ANDed together.

The schema for the XML document that is returned in the DETAILS column is available in the file `sqllib/misc/DB2MonRoutines.xsd`. Further details can be found in the file `sqllib/misc/DB2MonCommon.xsd`.

Information returned

Table 137. Information returned for MON_GET_UNIT_OF_WORK_DETAILS

Column Name	Data Type	Description or corresponding monitor element
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	service_superclass_name - Service superclass name
SERVICE_SUBCLASS_NAME	VARCHAR(128)	service_subclass_name - Service subclass name
SERVICE_CLASS_ID	INTEGER	service_class_id - Service class ID
MEMBER	SMALLINT	member - Database member
COORD_MEMBER	SMALLINT	coord_member - Coordinator member Database member for the coordinator partition of the given unit of work.
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
WORKLOAD_NAME	VARCHAR(128)	workload_name - Workload name
WORKLOAD_OCCURRENCE_ID	INTEGER	workload_occurrence_id - Workload occurrence identifier. This ID does not uniquely identify the workload occurrence unless it is coupled with the coordinator database partition number and the workload name.
UOW_ID	INTEGER	uow_id - Unit of work ID
DETAILS	BLOB(1M)	XML document that contains detailed metrics for the unit of work. See Table 138 on page 487 for a description of the elements in this document.

The following example shows the structure of the XML document that is returned in the DETAILS column.

```
<db2_unit_of_work xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="90700000">
   <service_superclass_name>SYSDEFAULTUSERCLASS</service_superclass_name>
   <service_subclass_name>SYSDEFAULTSUBCLASS</service_subclass_name>
   <service_class_id>13</service_class_id>
   <workload_name>SYSDEFAULTUSERWORKLOAD</workload_name>
   <member>0</member>
```

```
   <coord_member>0</coord_member>
   <application_handle>21</application_handle>
   <workload_occurrence_id>1</workload_occurrence_id>
   <uow_id>2</uow_id>
   <workload_occurrence_state>UOWEXEC</workload_occurrence_state>
   <system_metrics>
   <act_aborted_total>5</act_aborted_total>
   ...
   <wlm_queue_assignments_total>3</wlm_queue_assignments_total>
   </system_metrics>
</db2_unit_of_work_metrics>
```

For the full schema, see `sqllib/misc/DB2MonRoutines.xsd`.

Table 138. Detailed metrics returned for MON_GET_UNIT_OF_WORK_DETAILS

Element Name	Data Type	Description or corresponding monitor element
service_superclass_name	xs:string (128)	service_superclass_name - Service superclass name
service_subclass_name	xs:string (128)	service_subclass_name - Service subclass name
service_class_id	xs:nonNegativeInteger	service_class_id - Service class ID
workload_name	xs:string (128)	workload_name - Workload name
member	xs:nonNegativeInteger	member - Database member
coord_member	xs:nonNegativeInteger	coord_member - Coordinator member
application_handle	xs:nonNegativeInteger	application_handle - Application handle
application_id	xs:string	appl_id - Application ID
workload_occurrence_id	xs:nonNegativeInteger	workload_occurrence_id - Workload occurrence identifier This ID does not uniquely identify the workload occurrence unless it is coupled with the coordinator member and the workload name.
uow_id	xs:nonNegativeInteger	uow_id - Unit of work ID
workload_occurrence_state	xs:string	workload_occurrence_state - Workload occurrence state
client_userid	xs:string	CURRENT CLIENT_USERID special register
client_wrkstnname	xs:string	CURRENT CLIENT_WRKSTNNAME special register
client_applname	xs:string	CURRENT CLIENT_APPLNAME special register
client_acctng	xs:string	CURRENT CLIENT_ACCTNG special register
act_aborted_total	xs:nonNegativeInteger	act_aborted_total - Total aborted activities
act_completed_total	xs:nonNegativeInteger	act_completed_total - Total completed activities
act_rejected_total	xs:nonNegativeInteger	act_rejected_total - Total rejected activities
act_rqsts_total	xs:nonNegativeInteger	act_rqsts_total - Total activity requests
agent_wait_time	xs:nonNegativeInteger	agent_wait_time - Agent wait time
agent_waits_total	xs:nonNegativeInteger	agent_waits_total - Total agent waits
app_rqsts_completed_total	xs:nonNegativeInteger	app_rqsts_completed_total - Total application requests completed
audit_events_total	xs:nonNetagiveIngteger	audit_events_total - Total audit events
audit_subsystem_wait_time	xs:nonNegativeInteger	audit_subsystem_wait_time - Audit subsystem wait time
audit_subsystem_waits_total	xs:nonNegativeInteger	audit_subsystem_waits_total - Total audit subsystem waits

Table 138. Detailed metrics returned for MON_GET_UNIT_OF_WORK_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
audit_file_write_wait_time	xs:nonNegativeInteger	audit_file_write_wait_time - Audit file write wait time
audit_file_writes_total	xs:nonNegativeInteger	audit_file_writes_total - Total Audit files written
cat_cache_inserts	xs:nonNegativeInteger	cat_cache_inserts - Catalog cache inserts
cat_cache_lookups	xs:nonNegativeInteger	cat_cache_lookups - Catalog cache lookups
client_hostname	xs:string	client_hostname - Client hostname
client_idle_wait_time	xs:nonNegativeInteger	client_idle_wait_time - Client idle wait time
client_port_number	xs:nonNegativeInteger	client_port_number - Client port number
deadlocks	xs:nonNegativeInteger	deadlocks - Deadlocks detected
diaglog_writes_total	xs:nonNegativeInteger	diaglog_writes_total - Diag log total writes
diaglog_write_wait_time	xs:nonNegativeInteger	diaglog_write_wait_time - Diag log write time
direct_read_time	xs:nonNegativeInteger	direct_read_time - Direct read time
direct_write_time	xs:nonNegativeInteger	direct_write_time - Direct write time
direct_read_reqs	xs:nonNegativeInteger	direct_read_reqs - Direct read requests
direct_reads	xs:nonNegativeInteger	direct_reads - Direct reads from database
direct_write_reqs	xs:nonNegativeInteger	direct_write_reqs - Direct write requests
direct_writes	xs:nonNegativeInteger	direct_writes - Direct writes to database
fcm_recv_volume	xs:nonNegativeInteger	fcm_recv_volume - FCM recv volume
fcm_recv_wait_time	xs:nonNegativeInteger	fcm_recv_wait_time - FCM recv wait time
fcm_recvs_total	xs:nonNegativeInteger	fcm_recvs_total - FCM recvs total
fcm_message_recv_volume	xs:nonNegativeInteger	fcm_message_recv_volume - FCM message recv volume
fcm_message_recvs_total	xs:nonNegativeInteger	fcm_message_recvs_total - FCM message recvs total
fcm_message_recv_wait_time	xs:nonNegativeInteger	fcm_message_recv_wait_time - FCM message recv wait time
fcm_message_send_volume	xs:nonNegativeInteger	fcm_message_send_volume - FCM message send volume
fcm_message_send_wait_time	xs:nonNegativeInteger	fcm_message_send_wait_time - FCM message send wait time
fcm_message_sends_total	xs:nonNegativeInteger	fcm_message_sends_total - FCM message sends total
fcm_send_volume	xs:nonNegativeInteger	fcm_send_volume - FCM send volume
fcm_send_wait_time	xs:nonNegativeInteger	fcm_send_wait_time - FCM send wait time
fcm_sends_total	xs:nonNegativeInteger	fcm_sends_total - FCM sends total
fcm_tq_recv_wait_time	xs:nonNegativeInteger	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
fcm_tq_send_wait_time	xs:nonNegativeInteger	fcm_tq_send_wait_time - FCM tablequeue send wait time
fcm_tq_recv_volume	xs:nonNegativeInteger	fcm_tq_recv_volume - FCM tablequeue recv volume
fcm_tq_recvs_total	xs:nonNegativeInteger	fcm_tq_recvs_total - FCM tablequeue recvs total
fcm_tq_send_volume	xs:nonNegativeInteger	fcm_tq_send_volume - FCM tablequeue send volume
fcm_tq_sends_total	xs:nonNegativeInteger	fcm_tq_sends_total - FCM tablequeue send total
int_commits	xs:nonNegativeInteger	int_commits - Internal commits

Table 138. Detailed metrics returned for MON_GET_UNIT_OF_WORK_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
int_rollbacks	xs:nonNegativeInteger	int_rollbacks - Internal rollbacks
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of tablequeue buffers overflowed
ipc_recv_volume	xs:nonNegativeInteger	ipc_recv_volume - Interprocess communication recv volume
ipc_recv_wait_time	xs:nonNegativeInteger	ipc_recv_wait_time - Interprocess communication recv wait time
ipc_recvs_total	xs:nonNegativeInteger	ipc_recvs_total - Interprocess communication recvs total
ipc_send_volume	xs:nonNegativeInteger	ipc_send_volume - Interprocess communication send volume
ipc_send_wait_time	xs:nonNegativeInteger	ipc_send_wait_time - Interprocess communication send wait time
ipc_sends_total	xs:nonNegativeInteger	ipc_sends_total - Interprocess communication send total
last_executable_id	xs:hexBinary(32)	last_executable_id - Last executable identifier
last_request_type	xs:string(32)	last_request_type - Last request type
lock_escals	xs:nonNegativeInteger	lock_escals - Number of lock escalations
lock_timeouts	xs:nonNegativeInteger	lock_timeouts - Number of lock timeouts
lock_wait_time	xs:nonNegativeInteger	lock_wait_time - Time waited on locks
lock_waits	xs:nonNegativeInteger	lock_waits - Lock waits
log_buffer_wait_time	xs:nonNegativeInteger	log_buffer_wait_time - Log buffer wait time
log_disk_wait_time	xs:nonNegativeInteger	log_disk_wait_time - Log disk wait time
log_disk_waits_total	xs:nonNegativeInteger	log_disk_waits_total - Log disk waits total
num_locks_held	xs:nonNegativeInteger	locks_held - Locks held
num_lw_thresh_exceeded	xs:nonNegativeInteger	num_lw_thresh_exceeded - Number of thresholds exceeded
thresh_violations	xs:nonNegativeInteger	thresh_violations - Number of threshold violations
pkg_cache_inserts	xs:nonNegativeInteger	pkg_cache_inserts - Package cache inserts
pkg_cache_lookups	xs:nonNegativeInteger	pkg_cache_lookups - Package cache lookups
pool_data_l_reads	xs:nonNegativeInteger	pool_data_l_reads - Buffer pool data logical reads
pool_data_p_reads	xs:nonNegativeInteger	pool_data_p_reads - Buffer pool data physical reads
pool_data_writes	xs:nonNegativeInteger	pool_data_writes - Buffer pool data writes
pool_index_l_reads	xs:nonNegativeInteger	pool_index_l_reads - Buffer pool index logical reads
pool_index_p_reads	xs:nonNegativeInteger	pool_index_p_reads - Buffer pool index physical reads
pool_index_writes	xs:nonNegativeInteger	pool_index_writes - Buffer pool index writes
pool_read_time	xs:nonNegativeInteger	pool_read_time - Total buffer pool physical read time
pool_temp_data_l_reads	xs:nonNegativeInteger	pool_temp_data_l_reads - Buffer pool temporary data logical reads
pool_temp_data_p_reads	xs:nonNegativeInteger	pool_temp_data_p_reads - Buffer pool temporary data physical reads

Table 138. Detailed metrics returned for MON_GET_UNIT_OF_WORK_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
pool_temp_index_l_reads	xs:nonNegativeInteger	pool_temp_index_l_reads - Buffer pool temporary index logical reads
pool_temp_index_p_reads	xs:nonNegativeInteger	pool_temp_index_p_reads - Buffer pool temporary index physical reads
pool_temp_xda_l_reads	xs:nonNegativeInteger	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
pool_temp_xda_p_reads	xs:nonNegativeInteger	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
pool_write_time	xs:nonNegativeInteger	pool_write_time - Total buffer pool physical write time
pool_xda_l_reads	xs:nonNegativeInteger	pool_xda_l_reads - Buffer pool XDA data logical reads
pool_xda_p_reads	xs:nonNegativeInteger	pool_xda_p_reads - Buffer pool XDA data physical reads
pool_xda_writes	xs:nonNegativeInteger	pool_xda_writes - Buffer pool XDA data writes
num_log_buffer_full	xs:nonNegativeInteger	num_log_buffer_full - Number of full log buffers
rqsts_completed_total	xs:nonNegativeInteger	rqsts_completed_total - Total requests completed
rows_modified	xs:nonNegativeInteger	rows_modified - Rows modified
rows_read	xs:nonNegativeInteger	rows_read - Rows read
rows_returned	xs:nonNegativeInteger	rows_returned - Rows returned
session_auth_id	xs:string	session_auth_id - Session authorization ID
tcpip_recv_volume	xs:nonNegativeInteger	tcpip_recv_volume - TCP/IP received volume
tcpip_recv_wait_time	xs:nonNegativeInteger	tcpip_recv_wait_time - TCP/IP recv wait time
tcpip_recvs_total	xs:nonNegativeInteger	tcpip_recvs_total - TCP/IP recvs total
tcpip_send_volume	xs:nonNegativeInteger	tcpip_send_volume - TCP/IP send volume
tcpip_send_wait_time	xs:nonNegativeInteger	tcpip_send_wait_time - TCP/IP send wait time
tcpip_sends_total	xs:nonNegativeInteger	tcpip_sends_total - TCP/IP sends total
total_act_time	xs:nonNegativeInteger	total_act_time - Total activity time
total_act_wait_time	xs:nonNegativeInteger	total_act_wait_time - Total activity wait time
total_app_commits	xs:nonNegativeInteger	total_app_commits - Total application commits
total_app_rollbacks	xs:nonNegativeInteger	total_app_rollbacks - Total application rollbacks
total_app_rqst_time	xs:nonNegativeInteger	total_app_rqst_time - Total application request time
total_app_section_executions	xs:nonNegativeInteger	total_app_section_executions - Total section executions
total_commit_proc_time	xs:nonNegativeInteger	total_commit_proc_time - Total commits processing time
total_commit_time	xs:nonNegativeInteger	total_commit_time - Total commit time
total_compilations	xs:nonNegativeInteger	total_compilations - Total compilations
total_compile_proc_time	xs:nonNegativeInteger	total_compile_proc_time - Total compile processing time
total_compile_time	xs:nonNegativeInteger	total_compile_time - Total compile time
total_cpu_time	xs:nonNegativeInteger	total_cpu_time - Total CPU time

Table 138. Detailed metrics returned for MON_GET_UNIT_OF_WORK_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
total_implicit_compilations	xs:nonNegativeInteger	total_implicit_compilations - Total implicit complications
total_implicit_compile_ proc_time	xs:nonNegativeInteger	total_implicit_compile_proc_time - Total implicit compile processing time
total_implicit_compile_time	xs:nonNegativeInteger	total_implicit_compile_time - Total implicit compile time
total_loads	xs:nonNegativeInteger	total_loads - Total loads
total_load_proc_time	xs:nonNegativeInteger	total_load_proc_time - Total load processing time
total_load_time	xs:nonNegativeInteger	total_load_time - Total load time
total_reorgs	xs:nonNegativeInteger	total_reorgs - Total reorganizations
total_reorg_proc_time	xs:nonNegativeInteger	total_reorg_proc_time - Total reorganization processing time
total_reorg_time	xs:nonNegativeInteger	total_reorg_time - Total reorganization time
total_rollback_proc_time	xs:nonNegativeInteger	total_rollback_proc_time - Total rollback processing time
total_rollback_time	xs:nonNegativeInteger	total_rollback_time - Total rollback time
total_routine_invocations	xs:nonNegativeInteger	total_routine_invocations - Total routine invocations
total_routine_time	xs:nonNegativeInteger	total_routine_time - Total routine time
total_routine_user_code _proc_time	xs:nonNegativeInteger	total_routine_user_code_proc_time - Total routine user code processing time
total_routine_user_code_time	xs:nonNegativeInteger	total_routine_user_code_time - Total routine user code time
total_rqst_time	xs:nonNegativeInteger	total_rqst_time - Total request time
total_runstats	xs:nonNegativeInteger	total_runstats - Total runtime statistics
total_runstats_proc_time	xs:nonNegativeInteger	total_runstats_proc_time - Total runtime statistics processing time
total_runstats_time	xs:nonNegativeInteger	total_runstats_time - Total runtime statistics
total_section_proc_time	xs:nonNegativeInteger	total_section_proc_time - Total section processing time
total_section_sort_time	xs:nonNegativeInteger	total_section_sort_time - Total section sort time
total_section_sort_proc_time	xs:nonNegativeInteger	total_section_sort_proc_time - Total section sort processing time
total_section_sorts	xs:nonNegativeInteger	total_section_sorts - Total section sorts
total_section_time	xs:nonNegativeInteger	total_section_time - Total section time
total_sorts	xs:nonNegativeInteger	total_sorts - Total Sorts
post_threshold_sorts	xs:nonNegativeInteger	post_threshold_sorts - Post threshold sorts
post_shrthreshold_sorts	xs:nonNegativeInteger	post_shrthreshold_sorts - Post shared threshold sorts
sort_overflows	xs:nonNegativeInteger	sort_overflows - Sort overflows
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of table queue buffers overflowed
total_wait_time	xs:nonNegativeInteger	total_wait_time - Total wait time
uow_log_space_used	xs:nonNegativeInteger	uow_log_space_used - Unit of Work Log Space Used
uow_start_time	xs:dateTime	uow_start_time - Unit of Work Start Timestamp

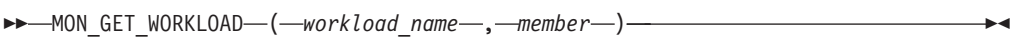
Table 138. Detailed metrics returned for MON_GET_UNIT_OF_WORK_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
wlm_queue_time_total	xs:nonNegativeInteger	wlm_queue_time_total - Workload manager total queue time
wlm_queue_assignments_total	xs:nonNegativeInteger	wlm_queue_assignments_total - Workload manager total queue assignments

MON_GET_WORKLOAD table function - Get workload metrics

The MON_GET_WORKLOAD table function returns metrics for one or more workloads.

Syntax



The schema is SYSPROC.

Table function parameters

workload_name
An input argument of type VARCHAR(128) that specifies a specific workload for which the metrics are to be returned. If the argument is NULL or an empty string, metrics are returned for all workloads.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the NULL value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_WORKLOAD function.

Example

Display lock information for each workload, aggregated across member, ordered by highest lock wait time.

```
SELECT varchar(workload_name,30) as workload_name,
       sum(lock_wait_time) as total_lock_wait_time,
       sum(lock_waits) as total_lock_waits,
       sum(lock_timeouts) as total_lock_timeouts,
       sum(lock_escals) as total_lock_escals
FROM TABLE(MON_GET_WORKLOAD('',-2)) AS t
GROUP BY workload_name
ORDER BY total_lock_wait_time DESC
```

The following is an example of output from this query.

```
WORKLOAD_NAME                  TOTAL_LOCK_WAIT_TIME TOTAL_LOCK_WAITS     ...
------------------------------ -------------------- -------------------- ...
SYSDEFAULTADMWORKLOAD                             0                    0 ...
SYSDEFAULTUSERWORKLOAD                            0                    0 ...

  2 record(s) selected.
```

Output for query (continued).

```
... TOTAL_LOCK_TIMEOUTS  TOTAL_LOCK_ESCALS
... -------------------- --------------------
...                    0                    0
...                    0                    0
```

Usage notes

The metrics returned by the MON_GET_WORKLOAD table function represent the accumulation of all metrics for requests that were submitted by connections mapped to the identified workload object. Metrics are rolled up to a workload on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_WORKLOAD table function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses and the *mon_req_metrics* database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default, request metrics are enabled at the database level. If request metrics have been disabled at the database level, and for a service superclass, then the metrics reported for each workload that is mapped to that service superclass stop increasing (or remain at 0 if request metrics were disabled at database activation time).

The MON_GET_WORKLOAD table function returns one row of data per workload and per member. No aggregation across workloads (on a member), or across members (for a service class or more), is performed. However, aggregation can be achieved through SQL queries (see the example).

Information returned

Table 139. Information returned for MON_GET_WORKLOAD

Column Name	Data Type	Description or corresponding monitor element
WORKLOAD_NAME	VARCHAR(128)	workload_name - Workload name
WORKLOAD_ID	INTEGER	workload_id - Workload ID
MEMBER	SMALLINT	member - Database member
ACT_ABORTED_TOTAL	BIGINT	act_aborted_total - Total aborted activities
ACT_COMPLETED_TOTAL	BIGINT	act_completed_total - Total completed activities
ACT_REJECTED_TOTAL	BIGINT	act_rejected_total - Total rejected activities
AGENT_WAIT_TIME	BIGINT	agent_wait_time - Agent wait time
AGENT_WAITS_TOTAL	BIGINT	agent_waits_total - Total agent waits
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads

Table 139. Information returned for MON_GET_WORKLOAD (continued)

Column Name	Data Type	Description or corresponding monitor element
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer pool XDA data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer pool XDA data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer pool XDA data writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
CLIENT_IDLE_WAIT_TIME	BIGINT	client_idle_wait_time - Client idle wait time
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
FCM_RECV_VOLUME	BIGINT	fcm_recv_volume - FCM recv volume
FCM_RECVS_TOTAL	BIGINT	fcm_recvs_total - FCM recvs total
FCM_SEND_VOLUME	BIGINT	fcm_send_volume - FCM send volume
FCM_SENDS_TOTAL	BIGINT	fcm_sends_total - FCM sends total
FCM_RECV_WAIT_TIME	BIGINT	fcm_recv_wait_time - FCM recv wait time
FCM_SEND_WAIT_TIME	BIGINT	fcm_send_wait_time - FCM send wait time
IPC_RECV_VOLUME	BIGINT	ipc_recv_volume - Interprocess communication recv volume
IPC_RECV_WAIT_TIME	BIGINT	ipc_recv_wait_time - Interprocess communication recv wait time
IPC_RECVS_TOTAL	BIGINT	ipc_recvs_total - Interprocess communication recvs total
IPC_SEND_VOLUME	BIGINT	ipc_send_volume - Interprocess communication send volume

Table 139. Information returned for MON_GET_WORKLOAD (continued)

Column Name	Data Type	Description or corresponding monitor element
IPC_SEND_WAIT_TIME	BIGINT	ipc_send_wait_time - Interprocess communication send wait time
IPC_SENDS_TOTAL	BIGINT	ipc_sends_total - Interprocess communication send total
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOG_BUFFER_WAIT_TIME	BIGINT	log_buffer_wait_time - Log buffer wait time
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers
LOG_DISK_WAIT_TIME	BIGINT	log_disk_wait_time - Log disk wait time
LOG_DISK_WAITS_TOTAL	BIGINT	log_disk_waits_total - Log disk waits total
RQSTS_COMPLETED_TOTAL	BIGINT	rqsts_completed_total - Total requests completed
ROWS_MODIFIED	BIGINT	rows_modified - Rows modified
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_RETURNED	BIGINT	rows_returned - Rows returned
TCPIP_RECV_VOLUME	BIGINT	tcpip_recv_volume - TCP/IP received volume
TCPIP_SEND_VOLUME	BIGINT	tcpip_send_volume - TCP/IP send volume
TCPIP_RECV_WAIT_TIME	BIGINT	tcpip_recv_wait_time - TCP/IP recv wait time
TCPIP_RECVS_TOTAL	BIGINT	tcpip_recvs_total - TCP/IP recvs total
TCPIP_SEND_WAIT_TIME	BIGINT	tcpip_send_wait_time - TCP/IP send wait time
TCPIP_SENDS_TOTAL	BIGINT	tcpip_sends_total - TCP/IP sends total
TOTAL_APP_RQST_TIME	BIGINT	total_app_rqst_time - Total application request time
TOTAL_RQST_TIME	BIGINT	total_rqst_time - Total request time
WLM_QUEUE_TIME_TOTAL	BIGINT	wlm_queue_time_total - Workload manager total queue time
WLM_QUEUE_ASSIGNMENTS _TOTAL	BIGINT	wlm_queue_assignments_total - Workload manager total queue assignments
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
TOTAL_WAIT_TIME	BIGINT	total_wait_time - Total wait time
APP_RQSTS_COMPLETED_TOTAL	BIGINT	app_rqsts_completed_total - Total application requests completed
TOTAL_SECTION_SORT_TIME	BIGINT	total_section_sort_time - Total section sort time
TOTAL_SECTION_SORT_PROC_TIME	BIGINT	total_section_sort_proc_time - Total section sort processing time
TOTAL_SECTION_SORTS	BIGINT	total_section_sorts - Total section sorts
TOTAL_SORTS	BIGINT	total_sorts - Total Sorts
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_COMPILE_TIME	BIGINT	total_compile_time - Total compile time

Table 139. Information returned for MON_GET_WORKLOAD (continued)

Column Name	Data Type	Description or corresponding monitor element
TOTAL_COMPILE_PROC_TIME	BIGINT	total_compile_proc_time - Total compile processing time
TOTAL_COMPILATIONS	BIGINT	total_compilations - Total compilations
TOTAL_IMPLICIT_COMPILE_TIME	BIGINT	total_implicit_compile_time - Total implicit compile time
TOTAL_IMPLICIT_COMPILE _PROC_TIME	BIGINT	total_implicit_compile_proc_time - Total implicit compile processing time
TOTAL_IMPLICIT_COMPILATIONS	BIGINT	total_implicit_compilations - Total implicit complications
TOTAL_SECTION_TIME	BIGINT	total_section_time - Total section time
TOTAL_SECTION_PROC_TIME	BIGINT	total_section_proc_time - Total section processing time
TOTAL_APP_SECTION_EXECUTIONS	BIGINT	total_app_section_executions - Total section executions
TOTAL_ACT_TIME	BIGINT	total_activity_time - Total activity time
TOTAL_ACT_WAIT_TIME	BIGINT	total_activity_wait_time - Total activity wait time
ACT_RQSTS_TOTAL	BIGINT	act_rqsts_total - Total activity requests
TOTAL_ROUTINE_TIME	BIGINT	total_routine_time - Total routine time
TOTAL_ROUTINE_INVOCATIONS	BIGINT	total_routine_invocations - Total routine invocations
TOTAL_COMMIT_TIME	BIGINT	total_commit_time - Total commit time
TOTAL_COMMIT_PROC_TIME	BIGINT	total_commit_proc_time - Total commits processing time
TOTAL_APP_COMMITS	BIGINT	total_app_commits - Total application commits
INT_COMMITS	BIGINT	int_commits - Internal commits
TOTAL_ROLLBACK_TIME	BIGINT	total_rollback_time - Total rollback time
TOTAL_ROLLBACK_PROC_TIME	BIGINT	total_rollback_proc_time - Total rollback processing time
TOTAL_APP_ROLLBACKS	BIGINT	total_app_rollbacks - Total application rollbacks
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
TOTAL_RUNSTATS_TIME	BIGINT	total_runstats_time - Total runtime statistics
TOTAL_RUNSTATS_PROC_TIME	BIGINT	total_runstats_proc_time - Total runtime statistics processing time
TOTAL_RUNSTATS	BIGINT	total_runstats - Total runtime statistics
TOTAL_REORG_TIME	BIGINT	total_reorg_time - Total reorganization time
TOTAL_REORG_PROC_TIME	BIGINT	total_reorg_proc_time - Total reorganization processing time
TOTAL_REORGS	BIGINT	total_reorgs - Total reorganizations
TOTAL_LOAD_TIME	BIGINT	total_load_time - Total load time
TOTAL_LOAD_PROC_TIME	BIGINT	total_load_proc_time - Total load processing time
TOTAL_LOADS	BIGINT	total_loads - Total loads
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts

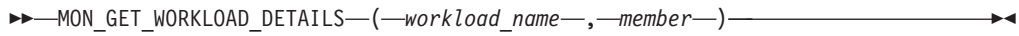
Table 139. Information returned for MON_GET_WORKLOAD (continued)

Column Name	Data Type	Description or corresponding monitor element
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
THRESH_VIOLATIONS	BIGINT	hresh_violations - Number of threshold violations
NUM_LW_THRESH_EXCEEDED	BIGINT	num_lw_thresh_exceeded - Number of thresholds exceeded
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

MON_GET_WORKLOAD_DETAILS table function - Get detailed workload metrics

The MON_GET_WORKLOAD_DETAILS table function returns detailed metrics for one or more workloads.

Syntax



The schema is SYSPROC.

Table function parameters

workload_name
An input argument of type VARCHAR(128) that specifies a specific workload for which the metrics are to be returned. If the argument is NULL or an empty string, metrics are returned for all workloads.

member
An input argument of type INTEGER that specifies a valid member in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the NULL value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the MON_GET_WORKLOAD_DETAILS function.

Example

Display lock information for each workload, aggregated across members, ordered by highest lock wait time.

```
SELECT varchar(wlmetrics.workload_name,30) as workload_name,
       sum(detmetrics.lock_wait_time) as total_lock_wait_time,
       sum(detmetrics.lock_waits) as total_lock_waits,
       sum(detmetrics.lock_timeouts) as total_lock_timeouts,
       sum(detmetrics.lock_escals) as total_lock_escals
FROM TABLE(MON_GET_WORKLOAD_DETAILS('',-2)) AS WLMETRICS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
           '$detmetric/db2_workload' PASSING
           XMLPARSE(DOCUMENT WLMETRICS.DETAILS)
           as "detmetric"
COLUMNS "LOCK_WAIT_TIME" INTEGER PATH 'system_metrics/lock_wait_time',
        "LOCK_WAITS" INTEGER PATH 'system_metrics/lock_waits',
        "LOCK_TIMEOUTS" INTEGER PATH 'system_metrics/lock_timeouts',
```

```
        "LOCK_ESCALS" INTEGER PATH 'system_metrics/lock_escals'
) AS DETMETRICS
GROUP BY workload_name
ORDER BY total_lock_wait_time desc;
```

The following is an example of output from this query.

```
WORKLOAD_NAME                  TOTAL_LOCK_WAIT_TIME TOTAL_LOCK_WAITS ...
------------------------------ -------------------- ---------------- ...
SYSDEFAULTADMWORKLOAD                             0                0 ...
SYSDEFAULTUSERWORKLOAD                            0                0 ...

  2 record(s) selected.
```

Output for query (continued).

```
... TOTAL_LOCK_TIMEOUTS TOTAL_LOCK_ESCALS
... ------------------- -----------------
...                   0                 0
...                   0                 0
```

Usage notes

The metrics returned by the MON_GET_WORKLOAD_DETAILS function represent the accumulation of all metrics for requests that were submitted by connections mapped to the identified workload object. This function is similar to the MON_GET_WORKLOAD table function:

- The MON_GET_WORKLOAD table function returns the most commonly used metrics in a column-based format and is the most performance efficient method of retrieving metrics.
- The MON_GET_WORKLOAD_DETAILS table function returns the entire set of available metrics in an XML document format, which provides maximum flexibility for formatting output. The XML based output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function (see the example).

Metrics are rolled up to a workload on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_WORKLOAD_DETAILS table function to query the metric at the start and end of the interval, and compute the difference.

Request metrics are controlled through the COLLECT REQUEST METRICS clause on service superclasses and the **mon_req_metrics** database configuration parameter at the database level. Metrics are only collected for a request if the request is processed by an agent in a service subclass whose parent service superclass has request metrics enabled, or if request metrics collection is enabled for the entire database. By default request metrics are enabled at the database level. If request metrics have been disabled at the database level, and for a service superclass, the metrics reported for each workload mapped to that service superclass stop increasing (or remain at 0 if request metrics were disabled at database activation time).

The MON_GET_WORKLOAD_DETAILS table function returns one row of data per workload and per member. No aggregation across workloads (on a member), or across members (for a service class or more), is performed. However, aggregation can be achieved through SQL queries as shown in the example.

The schema for the XML document that is returned in the DETAILS column is available in the file sqllib/misc/DB2MonRoutines.xsd. Further details can be found in the file sqllib/misc/DB2MonCommon.xsd.

Information returned

Table 140. Information returned for MON_GET_WORKLOAD_DETAILS

Column Name	Data Type	Description
WORKLOAD_NAME	VARCHAR(128)	workload_name - Workload name
WORKLOAD_ID	INTEGER	workload_id - Workload ID
MEMBER	SMALLINT	member - Database member
DETAILS	BLOB(1M)	XML document that contains detailed metrics for the workload. See Table 141 for a description of the elements in this document.

The following example shows the structure of the XML document that is returned in the DETAILS column.

```
<db2_workload xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="90700000">
   <workload_name>SYSDEFAULTADMWORKLOAD</workload_name>
   <workload_id>11</workload_id>
   <member>0</member>
   <system_metrics release="90700000">
   <act_aborted_total>5</act_aborted_total>
    ...
   <wlm_queue_assignments_total>3</wlm_queue_assignments_total>
   </system_metrics>
</db2_workload>
```

For the full schema, see sqllib/misc/DB2MonRoutines.xsd.

Table 141. Detailed metrics returned for MON_GET_WORKLOAD_DETAILS

Element Name	Data Type	Description or corresponding monitor element
workload_name	xs:string (128)	workload_name - Workload name
workload_id	xs:nonNegativeInteger	workload_id - Workload ID
member	xs:nonNegativeInteger	member - Database member
act_aborted_total	xs:nonNegativeInteger	act_aborted_total - Total aborted activities
act_completed_total	xs:nonNegativeInteger	act_completed_total - Total completed activities
act_rejected_total	xs:nonNegativeInteger	act_rejected_total - Total rejected activities
act_rqsts_total	xs:nonNegativeInteger	act_rqsts_total - Total activity requests
agent_wait_time	xs:nonNegativeInteger	agent_wait_time - Agent wait time
agent_waits_total	xs:nonNegativeInteger	agent_waits_total - Total agent waits
app_rqsts_completed_total	xs:nonNegativeInteger	app_rqsts_completed_total - Total application requests completed
audit_events_total	xs:nonNetagiveIngteger	audit_events_total - Total audit events
audit_subsystem_wait_time	xs:nonNegativeInteger	audit_subsystem_wait_time - Audit subsystem wait time
audit_subsystem_waits_total	xs:nonNegativeInteger	audit_subsystem_waits_total - Total audit subsystem waits
audit_file_write_wait_time	xs:nonNegativeInteger	audit_file_write_wait_time - Audit file write wait time
audit_file_writes_total	xs:nonNegativeInteger	audit_file_writes_total - Total Audit files written
cat_cache_inserts	xs:nonNegativeInteger	cat_cache_inserts - Catalog cache inserts

Table 141. Detailed metrics returned for MON_GET_WORKLOAD_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
cat_cache_lookups	xs:nonNegativeInteger	cat_cache_lookups - Catalog cache lookups
client_idle_wait_time	xs:nonNegativeInteger	client_idle_wait_time - Client idle wait time
deadlocks	xs:nonNegativeInteger	deadlocks - Deadlocks detected
diaglog_writes_total	xs:nonNegativeInteger	diaglog_writes_total - Diag log total writes
diaglog_write_wait_time	xs:nonNegativeInteger	diaglog_write_wait_time - Diag log write time
direct_read_time	xs:nonNegativeInteger	direct_read_time - Direct read time
direct_write_time	xs:nonNegativeInteger	direct_write_time - Direct write time
direct_read_reqs	xs:nonNegativeInteger	direct_read_reqs - Direct read requests
direct_reads	xs:nonNegativeInteger	direct_reads - Direct reads from database
direct_write_reqs	xs:nonNegativeInteger	direct_write_reqs - Direct write requests
direct_writes	xs:nonNegativeInteger	direct_writes - Direct writes to database
fcm_recv_volume	xs:nonNegativeInteger	fcm_recv_volume - FCM recv volume
fcm_recv_wait_time	xs:nonNegativeInteger	fcm_recv_wait_time - FCM recv wait time
fcm_recvs_total	xs:nonNegativeInteger	fcm_recvs_total - FCM recvs total
fcm_message_recv_volume	xs:nonNegativeInteger	fcm_message_recv_volume - FCM message recv volume
fcm_message_recvs_total	xs:nonNegativeInteger	fcm_message_recvs_total - FCM message recvs total
fcm_message_recv_wait_time	xs:nonNegativeInteger	fcm_message_recv_wait_time - FCM message recv wait time
fcm_message_send_volume	xs:nonNegativeInteger	fcm_message_send_volume - FCM message send volume
fcm_message_send_wait_time	xs:nonNegativeInteger	fcm_message_send_wait_time - FCM message send wait time
fcm_message_sends_total	xs:nonNegativeInteger	fcm_message_sends_total - FCM message sends total
fcm_send_volume	xs:nonNegativeInteger	fcm_send_volume - FCM send volume
fcm_send_wait_time	xs:nonNegativeInteger	fcm_send_wait_time - FCM send wait time
fcm_sends_total	xs:nonNegativeInteger	fcm_sends_total - FCM sends total
fcm_tq_recv_wait_time	xs:nonNegativeInteger	fcm_tq_recv_wait_time - FCM tablequeue recv wait time
fcm_tq_send_wait_time	xs:nonNegativeInteger	fcm_tq_send_wait_time - FCM tablequeue send wait time
fcm_tq_recv_volume	xs:nonNegativeInteger	fcm_tq_recv_volume - FCM tablequeue recv volume
fcm_tq_recvs_total	xs:nonNegativeInteger	fcm_tq_recvs_total - FCM tablequeue recvs total
fcm_tq_send_volume	xs:nonNegativeInteger	fcm_tq_send_volume - FCM tablequeue send volume
fcm_tq_sends_total	xs:nonNegativeInteger	fcm_tq_sends_total - FCM tablequeue send total
int_commits	xs:nonNegativeInteger	int_commits - Internal commits
int_rollbacks	xs:nonNegativeInteger	int_rollbacks - Internal rollbacks
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of tablequeue buffers overflowed
ipc_recv_volume	xs:nonNegativeInteger	ipc_recv_volume - Interprocess communication recv volume
ipc_recv_wait_time	xs:nonNegativeInteger	ipc_recv_wait_time - Interprocess communication recv wait time
ipc_recvs_total	xs:nonNegativeInteger	ipc_recvs_total - Interprocess communication recvs total

Table 141. Detailed metrics returned for MON_GET_WORKLOAD_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
ipc_send_volume	xs:nonNegativeInteger	ipc_send_volume - Interprocess communication send volume
ipc_send_wait_time	xs:nonNegativeInteger	ipc_send_wait_time - Interprocess communication send wait time
ipc_sends_total	xs:nonNegativeInteger	ipc_sends_total - Interprocess communication send total
lock_escals	xs:nonNegativeInteger	lock_escals - Number of lock escalations
lock_timeouts	xs:nonNegativeInteger	lock_timeouts - Number of lock timeouts
lock_wait_time	xs:nonNegativeInteger	lock_wait_time - Time waited on locks
lock_waits	xs:nonNegativeInteger	lock_waits - Lock waits
log_buffer_wait_time	xs:nonNegativeInteger	log_buffer_wait_time - Log buffer wait time
log_disk_wait_time	xs:nonNegativeInteger	log_disk_wait_time - Log disk wait time
log_disk_waits_total	xs:nonNegativeInteger	log_disk_waits_total - Log disk waits total
num_lw_thresh_exceeded	xs:nonNegativeInteger	num_lw_thresh_exceeded - Number of thresholds exceeded
pkg_cache_inserts	xs:nonNegativeInteger	pkg_cache_inserts - Package cache inserts
pkg_cache_lookups	xs:nonNegativeInteger	pkg_cache_lookups - Package cache lookups
pool_data_l_reads	xs:nonNegativeInteger	pool_data_l_reads - Buffer pool data logical reads
pool_data_p_reads	xs:nonNegativeInteger	pool_data_p_reads - Buffer pool data physical reads
pool_data_writes	xs:nonNegativeInteger	pool_data_writes - Buffer pool data writes
pool_index_l_reads	xs:nonNegativeInteger	pool_index_l_reads - Buffer pool index logical reads
pool_index_p_reads	xs:nonNegativeInteger	pool_index_p_reads - Buffer pool index physical reads
pool_index_writes	xs:nonNegativeInteger	pool_index_writes - Buffer pool index writes
pool_read_time	xs:nonNegativeInteger	pool_read_time - Total buffer pool physical read time
pool_temp_data_l_reads	xs:nonNegativeInteger	pool_temp_data_l_reads - Buffer pool temporary data logical reads
pool_temp_data_p_reads	xs:nonNegativeInteger	pool_temp_data_p_reads - Buffer pool temporary data physical reads
pool_temp_index_l_reads	xs:nonNegativeInteger	pool_temp_index_l_reads - Buffer pool temporary index logical reads
pool_temp_index_p_reads	xs:nonNegativeInteger	pool_temp_index_p_reads - Buffer pool temporary index physical reads
pool_temp_xda_l_reads	xs:nonNegativeInteger	pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads
pool_temp_xda_p_reads	xs:nonNegativeInteger	pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
pool_write_time	xs:nonNegativeInteger	pool_write_time - Total buffer pool physical write time
pool_xda_l_reads	xs:nonNegativeInteger	pool_xda_l_reads - Buffer pool XDA data logical reads
pool_xda_p_reads	xs:nonNegativeInteger	pool_xda_p_reads - Buffer pool XDA data physical reads
pool_xda_writes	xs:nonNegativeInteger	pool_xda_writes - Buffer pool XDA data writes
num_log_buffer_full	xs:nonNegativeInteger	num_log_buffer_full - Number of full log buffers
rqsts_completed_total	xs:nonNegativeInteger	rqsts_completed_total - Total requests completed
rows_modified	xs:nonNegativeInteger	rows_modified - Rows modified
rows_read	xs:nonNegativeInteger	rows_read - Rows read

Table 141. Detailed metrics returned for MON_GET_WORKLOAD_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
rows_returned	xs:nonNegativeInteger	rows_returned - Rows returned
tcpip_recv_volume	xs:nonNegativeInteger	tcpip_recv_volume - TCP/IP received volume
tcpip_recv_wait_time	xs:nonNegativeInteger	tcpip_recv_wait_time - TCP/IP recv wait time
tcpip_recvs_total	xs:nonNegativeInteger	tcpip_recvs_total - TCP/IP recvs total
tcpip_send_volume	xs:nonNegativeInteger	tcpip_send_volume - TCP/IP send volume
tcpip_send_wait_time	xs:nonNegativeInteger	tcpip_send_wait_time - TCP/IP send wait time
tcpip_sends_total	xs:nonNegativeInteger	tcpip_sends_total - TCP/IP sends total
thresh_violations	xs:nonNegativeInteger	thresh_violations - Number of threshold violations
total_act_time	xs:nonNegativeInteger	total_act_time - Total activity time
total_act_wait_time	xs:nonNegativeInteger	total_act_wait_time - Total activity wait time
total_app_commits	xs:nonNegativeInteger	total_app_commits - Total application commits
total_app_rollbacks	xs:nonNegativeInteger	total_app_rollbacks - Total application rollbacks
total_app_rqst_time	xs:nonNegativeInteger	total_app_rqst_time - Total application request time
total_app_section_executions	xs:nonNegativeInteger	total_app_section_executions - Total section executions
total_commit_proc_time	xs:nonNegativeInteger	total_commit_proc_time - Total commits processing time
total_commit_time	xs:nonNegativeInteger	total_commit_time - Total commit time
total_compilations	xs:nonNegativeInteger	total_compilations - Total compilations
total_compile_proc_time	xs:nonNegativeInteger	total_compile_proc_time - Total compile processing time
total_compile_time	xs:nonNegativeInteger	total_compile_time - Total compile time
total_cpu_time	xs:nonNegativeInteger	total_cpu_time - Total CPU time
total_implicit_compilations	xs:nonNegativeInteger	total_implicit_compilations - Total implicit complications
total_implicit_compile_ proc_time	xs:nonNegativeInteger	total_implicit_compile_proc_time - Total implicit compile processing time
total_implicit_compile_time	xs:nonNegativeInteger	total_implicit_compile_time - Total implicit compile time
total_loads	xs:nonNegativeInteger	total_loads - Total loads
total_load_proc_time	xs:nonNegativeInteger	total_load_proc_time - Total load processing time
total_load_time	xs:nonNegativeInteger	total_load_time - Total load time
total_reorgs	xs:nonNegativeInteger	total_reorgs - Total reorganizations
total_reorg_proc_time	xs:nonNegativeInteger	total_reorg_proc_time - Total reorganization processing time
total_reorg_time	xs:nonNegativeInteger	total_reorg_time - Total reorganization time
total_rollback_proc_time	xs:nonNegativeInteger	total_rollback_proc_time - Total rollback processing time
total_rollback_time	xs:nonNegativeInteger	total_rollback_time - Total rollback time
total_routine_invocations	xs:nonNegativeInteger	total_routine_invocations - Total routine invocations
total_routine_time	xs:nonNegativeInteger	total_routine_time - Total routine time
total_routine_user_ code_proc_time	xs:nonNegativeInteger	total_routine_user_code_proc_time - Total routine user code processing time
total_routine_user_ code_time	xs:nonNegativeInteger	total_routine_user_code_time - Total routine user code time
total_rqst_time	xs:nonNegativeInteger	total_rqst_time - Total request time
total_runstats	xs:nonNegativeInteger	total_runstats - Total runtime statistics

Table 141. Detailed metrics returned for MON_GET_WORKLOAD_DETAILS (continued)

Element Name	Data Type	Description or corresponding monitor element
total_runstats_proc_time	xs:nonNegativeInteger	total_runstats_proc_time - Total runtime statistics processing time
total_runstats_time	xs:nonNegativeInteger	total_runstats_time - Total runtime statistics
total_section_proc_time	xs:nonNegativeInteger	total_section_proc_time - Total section processing time
total_section_sort_time	xs:nonNegativeInteger	total_section_sort_time - Total section sort time
total_section_sort_proc_time	xs:nonNegativeInteger	total_section_sort_proc_time - Total section sort processing time
total_section_sorts	xs:nonNegativeInteger	total_section_sorts - Total section sorts
total_section_time	xs:nonNegativeInteger	total_section_time - Total section time
total_sorts	xs:nonNegativeInteger	total_sorts - Total Sorts
post_threshold_sorts	xs:nonNegativeInteger	post_threshold_sorts - Post threshold sorts
post_shrthreshold_sorts	xs:nonNegativeInteger	post_shrthreshold_sorts - Post shared threshold sorts
sort_overflows	xs:nonNegativeInteger	sort_overflows - Sort overflows
tq_tot_send_spills	xs:nonNegativeInteger	tq_tot_send_spills - Total number of table queue buffers overflowed
total_wait_time	xs:nonNegativeInteger	total_wait_time - Total wait time
wlm_queue_time_total	xs:nonNegativeInteger	wlm_queue_time_total - Workload manager total queue time
wlm_queue_assignments_total	xs:nonNegativeInteger	wlm_queue_assignments_total - Workload manager total queue assignments

MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks

The MON_LOCKWAITS administrative view returns information about agents working on behalf of applications that are waiting to obtain locks in the currently connected database. It is a useful query for identifying locking problems. This administrative view replaces the SNAPLOCKWAIT administrative view which is deprecated in DB2 Version 9.7 Fix Pack 1 and might be discontinued in a future release.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_LOCKWAITS administrative view
- CONTROL privilege on the MON_LOCKWAITS administrative view
- DATAACCESS authority

Information returned

Table 142. Information returned by the MON_LOCKWAITS administrative view

Column name	Data type	Description or Monitor element
LOCK_WAIT_ELAPSED _TIME	INTEGER	The time elapsed since the agent started waiting to obtain the lock. This value is given in seconds.
LOCK_NAME	VARCHAR(32)	lock_name - Lock name You can use the MON_FORMAT_LOCK_ NAME routine to format this internal binary lock name and obtain more details regarding the lock, such as the table and table space that a table lock references.
LOCK_OBJECT_TYPE	VARCHAR(32)	lock_object_type - Lock object type waited on
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name For locks that do not reference a table, NULL is returned.
TABNAME	VARCHAR(128)	table_name - Table name For locks that do not reference a table, NULL is returned.
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier This element is only applicable to partitioned tables and partitioned indexes. When returning lock level information, a value of -1 represents a lock which controls access to the whole table.
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode
LOCK_CURRENT_MODE	VARCHAR(10)	lock_current_mode - Original lock mode before conversion If the LOCK_STATUS is not "C" (converting), then a value of NULL is returned.
LOCK_MODE_REQUESTED	VARCHAR(10)	lock_mode_requested - Lock mode requested
REQ_APPLICATION_ HANDLE	BIGINT	req_application_handle - Requesting application handle

Table 142. Information returned by the MON_LOCKWAITS administrative view (continued)

Column name	Data type	Description or Monitor element
REQ_AGENT_TID	BIGINT	req_agent_tid - Requesting agent TID
REQ_MEMBER	SMALLINT	req_member - Requesting member
REQ_APPLICATION_ NAME	VARCHAR(128)	The name of the application running at the client that is waiting to acquire this lock.
REQ_USERID	VARCHAR(128)	The current authorization ID for the session being used by the application that is waiting to acquire this lock.
REQ_STMT_TEXT	CLOB(2MB)	SQL statement section that the application waiting to acquire the lock is executing. For non-SQL activities, a 0-length string value is returned.
HLD_APPLICATION_ HANDLE	BIGINT	hld_application_handle - Holding application handle If the application holding this lock is unknown or cannot be found then a value of NULL is returned.
HLD_MEMBER	SMALLINT	hld_member - Holding member
HLD_APPLICATION_ NAME	VARCHAR(128)	The name of the application running at the client that is holding this lock. If the application holding this lock is unknown or cannot be found then a 0-length string value is returned.
HLD_USERID	VARCHAR(128)	The current authorization ID for the session being used by the application that is holding this lock.
HLD_CURRENT_ STMT_TEXT	CLOB(2MB)	SQL statement text that is currently associated with the application that is holding the lock. Note that this is not necessarily the statement that is causing the lock.

MON_PKG_CACHE_SUMMARY - Retrieve a high-level summary of the database package cache

The MON_PKG_CACHE_SUMMARY administrative view returns key metrics for both static and dynamic SQL statements in the cache, providing a high-level summary of the database package cache. The metrics returned are aggregated over all executions of the statement across all members of the database.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_PKG_CACHE_SUMMARY administrative view
- CONTROL privilege on the MON_PKG_CACHE_SUMMARY administrative view
- DATAACCESS authority

Information returned

Table 143. Information returned by the MON_PKG_CACHE_SUMMARY administrative view

Column name	Data type	Description or Monitor element
SECTION_TYPE	CHAR(1)	section_type - Section type indicator
EXECUTABLE_ID	VARCHAR(32) FOR BIT DATA	executable_id - Executable ID
NUM_COORD_EXEC	BIGINT	num_coord_exec - Number of executions by coordinator agent
NUM_COORD_EXEC_ WITH_METRICS	BIGINT	num_coord_exec_with_metrics - Number of executions by coordinator agent with metrics
TOTAL_STMT_EXEC_TIME	BIGINT	The total amount of time, in milliseconds, spent executing the statement, including nested activities, over all executions of the statement where the metrics have been collected.
AVG_STMT_EXEC_TIME	BIGINT	The average amount of time, in milliseconds, spent executing the statement, including nested activities, over all executions of the statement where the metrics have been collected.

Table 143. Information returned by the MON_PKG_CACHE_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
TOTAL_CPU_TIME	BIGINT	The total amount of CPU time, in microseconds, used while within the DB2 database manager. This value represents the combined total of both user and system CPU time. It is calculated as the sum of all total_cpu_time - Total CPU time values for the statement.
AVG_CPU_TIME	BIGINT	The average amount of CPU time, in microseconds, spent within the DB2 database manager over all executions of the statement where the metrics have been collected.
TOTAL_LOCK_WAIT_TIME	BIGINT	The total elapsed time, in milliseconds, spent waiting for locks. This value is calculated as the sum of all lock_wait_time - Time waited on locks values for the statement.
AVG_LOCK_WAIT_TIME	BIGINT	The average elapsed time, in milliseconds, spent waiting for locks over all executions of the statement where the metrics have been collected.
TOTAL_IO_WAIT_TIME	BIGINT	The total elapsed time, in milliseconds, spent on I/O operations. This value is calculated as the sum of the elapsed time required to perform direct reads or direct writes, plus the elapsed time spent physically reading or writing data and index pages from or to the table space containers.
AVG_IO_WAIT_TIME	BIGINT	The average elapsed time, in milliseconds, spent on I/O operations over all executions of the statement where the metrics have been collected.
PREP_TIME	BIGINT	prep_time - Preparation time
ROWS_READ_PER_ ROWS_RETURNED	BIGINT	The average number of rows read per rows returned over all executions of the statement where the metrics have been collected.

Table 143. Information returned by the MON_PKG_CACHE_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
STMT_TEXT	CLOB(2MB)	stmt_text - SQL statement text

MON_SERVICE_SUBCLASS_SUMMARY - Retrieve metrics for all service subclasses

The MON_SERVICE_SUBCLASS_SUMMARY administrative view returns key metrics for all service subclasses in the currently connected database. It is designed to help monitor the system in a high-level manner, showing work executed per service class.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

The metrics returned represent the accumulation of all metrics for requests that have executed under the indicated service subclass across all members of the database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_SERVICE_SUBCLASS_SUMMARY administrative view
- CONTROL privilege on the MON_SERVICE_SUBCLASS_SUMMARY administrative view
- DATAACCESS authority

Information returned

Table 144. Information returned by the MON_SERVICE_SUBCLASS_SUMMARY administrative view

Column name	Data type	Description or Monitor element
SERVICE_SUPERCLASS _NAME	VARCHAR(128)	service_superclass_name - Service superclass name
SERVICE_SUBCLASS_NAME	VARCHAR(128)	service_subclass_name - Service subclass name
SERVICE_CLASS_ID	INTEGER	service_class_id - Service class ID
TOTAL_APP_COMMITS	BIGINT	Total number of application commits across all members of the database for the specified service subclass.

Table 144. Information returned by the MON_SERVICE_SUBCLASS_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
TOTAL_APP_ROLLBACKS	BIGINT	Total number of application rollbacks across all members of the database for the specified service subclass.
ACT_COMPLETED_TOTAL	BIGINT	Total number of coordinator activities at any nesting level that completed successfully across all members of the database for the specified service subclass.
APP_RQSTS_COMPLETED _TOTAL	BIGINT	Total number of external (application) requests that completed successfully across all members of the database for the specified service subclass
AVG_RQST_CPU_TIME	BIGINT	Average amount of CPU time, in microseconds, used by all external requests that completed successfully. It represents the total of both user and system CPU time.
ROUTINE_TIME_ RQST_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent working on requests that was spent executing user routines.
RQST_WAIT_ TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent working on requests that was spent waiting within the DB2 database server.
ACT_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of the time spent executing activities that was spent waiting within the DB2 database server.
IO_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent waiting within the DB2 database server that was due to I/O operations. This includes time spent performing direct reads or direct writes, and time spent reading data and index pages from the table space to the bufferpool or writing them back to disk.
LOCK_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent waiting on locks.
AGENT_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent by an application queued to wait for an agent under concentrator configurations.

Table 144. Information returned by the MON_SERVICE_SUBCLASS_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
NETWORK_WAIT_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent on client-server communications. This includes time spent sending and receiving data over TCP/IP or using the IPC protocol.
SECTION_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent executing sections. This includes the time spent performing sorts.
SECTION_SORT_ PROC_TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing sorts while executing sections.
COMPILE_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent compiling an SQL statement. This includes explicit and implicit compile times.
TRANSACT_END_PROC _TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing commit processing or rolling back transactions.
UTILS_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent running utilities. This includes performing runstats, reorganization, and load operations.
AVG_LOCK_WAITS _PER_ACT	BIGINT	The average number of times that applications or connections waited for locks per coordinator activities (successful and aborted).
AVG_LOCK_TIMEOUTS _PER_ACT	BIGINT	The average number of times that a request to lock an object timed out per coordinator activities (successful and aborted).
AVG_DEADLOCKS_ PER_ACT	BIGINT	The average number of deadlocks per coordinator activities (successful and aborted).

Table 144. Information returned by the MON_SERVICE_SUBCLASS_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
AVG_LOCK_ESCALS _PER_ACT	BIGINT	The average number of times that locks have been escalated from several row locks to a table lock per coordinator activities (successful and aborted).
ROWS_READ_PER_ ROWS_RETURNED	BIGINT	The average number of rows read from the table per rows returned to the application.
TOTAL_BP_HIT_ RATIO_PERCENT	DECIMAL(5,2)	The percentage of time that the database manager did not need to load a page from disk to service a data or index page request, including requests for XML storage objects (XDAs).

MON_TBSP_UTILIZATION - Retrieve monitoring metrics for all table spaces and all database partitions

The MON_TBSP_UTILIZATION administrative view returns key monitoring metrics, including hit ratios and utilization percentage, for all table spaces and all database partitions in the currently connected database. It provides critical information for monitoring performance as well as space utilization. This administrative view is a replacement for the TBSP_UTILIZATION administrative view.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_TBSP_UTILIZATION administrative view
- CONTROL privilege on the MON_TBSP_UTILIZATION administrative view
- DATAACCESS authority

Information returned

Table 145. Information returned by the MON_TBSP_UTILIZATION administrative view

Column name	Data type	Description or Monitor element
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
MEMBER	SMALLINT	member - Database member

Table 145. Information returned by the MON_TBSP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
TBSP_TYPE	VARCHAR(10)	tablespace_type - Table space type. This interface returns a text identifier based on defines in sqlutil.h, and is one of: • DMS • SMS
TBSP_CONTENT_TYPE	VARCHAR(10)	tablespace_content_type - Table space content type. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ANY • LARGE • SYSTEMP • USRTEMP
TBSP_STATE	VARCHAR(256)	tablespace_state - Table space state
TBSP_PAGE_SIZE	BIGINT	tablespace_page_size - Table space page size
TBSP_EXTENT_SIZE	BIGINT	tablespace_extent_size - Table space extent size
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size
TBSP_USING_ AUTO_STORAGE	SMALLINT	tablespace_using_auto _storage - Table space enabled for automatic storage
TBSP_AUTO_ RESIZE_ENABLED	SMALLINT	tablespace_auto_resize _enabled - Table space automatic resizing enabled
TBSP_TOTAL_SIZE_KB	BIGINT	The total size of the table space in kilobytes. This is calculated as (*tablespace_total_pages* * *tablespace_page_size*) / 1024 where *tablespace_total_pages* and *tablespace_page_size* represent the following monitor elements: • tablespace_total_pages - Total pages in table space • tablespace_page_size - Table space page size

Table 145. Information returned by the MON_TBSP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
TBSP_USABLE_SIZE_KB	BIGINT	The total usable size of the table space, in kilobytes. This equals the total size of the table space minus the space used for overhead pages. This is calculated as (*tablespace_usable_pages* * *tablespace_page_size*) / 1024 where *tablespace_usable_pages* and *tablespace_page_size* represent the following monitor elements: • tablespace_usable_pages - Usable pages in table space • tablespace_page_size - Table space page size
TBSP_UTILIZATION _PERCENT	DECIMAL(5,2)	The utilization of the table space as a percentage. If *tablespace_usable_pages* is greater than zero, this is calculated as (*tablespace_used_pages* / *tablespace_usable_pages*) * 100 where *tablespace_used_pages* and *tablespace_usable_pages* represent the following monitor elements: • tablespace_used_pages - Used pages in table space • tablespace_usable_pages - Usable pages in table space NULL is returned if *tablespace_usable_pages* is not greater than zero.
TBSP_PAGE_TOP	BIGINT	tablespace_page_top - Table space high watermark

Table 145. Information returned by the MON_TBSP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
DATA_PHYSICAL_READS	BIGINT	Indicates the number of data pages read from the table space containers (physical) for temporary as well as regular and large table spaces. This is calculated as (*pool_data_p_reads* + *pool_temp_data_p_reads*) where *pool_data_p_reads* and *pool_temp_data_p_reads* represent the following monitor elements: • pool_data_p_reads - Buffer pool data physical reads • pool_temp_data_p_reads - Buffer pool temporary data physical reads
DATA_HIT_ RATIO_PERCENT	DECIMAL(5,2)	Data hit ratio, that is, the percentage of time that the database manager did not need to load a page from disk to service a data page request.
INDEX_PHYSICAL_READS	BIGINT	Indicates the number of index pages read from the table space containers (physical) for temporary as well as regular and large table spaces. This is calculated as (*pool_index_p_reads* + *pool_temp_index_p_reads*) where *pool_index_p_reads* and *pool_temp_index_p_reads* represent the following monitor elements: • pool_index_p_reads - Buffer pool index physical reads • pool_temp_index_p_reads - Buffer pool temporary index physical reads
INDEX_HIT_RATIO _PERCENT	DECIMAL(5,2)	Index hit ratio, that is, the percentage of time that the database manager did not need to load a page from disk to service an index data page request.

Table 145. Information returned by the MON_TBSP_UTILIZATION administrative view (continued)

Column name	Data type	Description or Monitor element
XDA_PHYSICAL_READS	BIGINT	Indicates the number of data pages for XML storage objects (XDAs) read from the table space containers (physical) for temporary as well as regular and large table spaces. This is calculated as (*pool_xda_p_reads* + *pool_temp_xda_p_reads*) where *pool_xda_p_reads* and *pool_temp_xda_p_reads* represent the following monitor elements: • pool_xda_p_reads - Buffer pool XDA data physical reads • pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads
XDA_HIT_RATIO_PERCENT	DECIMAL(5,2)	Auxiliary storage objects hit ratio, that is, the percentage of time that the database manager did not need to load a page from disk to service a data page request for XML storage objects (XDAs).

MON_WORKLOAD_SUMMARY - Retrieves metrics for all workloads

The MON_WORKLOAD_SUMMARY administrative view returns key metrics for all workloads in the currently connected database. It is designed to help monitor the system in a high-level manner, showing incoming work per workload.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7, this routine will no longer work.

The metrics returned represent the accumulation of all metrics for requests that were submitted by connections mapped to the identified workload object across all members of the database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the MON_WORKLOAD_SUMMARY administrative view

- CONTROL privilege on the MON_WORKLOAD_SUMMARY administrative view
- DATAACCESS authority

Information returned

Table 146. Information returned by the MON_WORKLOAD_SUMMARY administrative view

Column name	Data type	Description or Monitor element
WORKLOAD_NAME	VARCHAR(128)	workload_name - Workload name
WORKLOAD_ID	INTEGER	workload_id - Workload ID
TOTAL_APP_COMMITS	BIGINT	Total number of application commits across all members of the database for the specified service subclass.
TOTAL_APP_ROLLBACKS	BIGINT	Total number of application rollbacks across all members of the database for the specified service subclass.
ACT_COMPLETED_TOTAL	BIGINT	Total number of coordinator activities at any nesting level that completed successfully across all members of the database for the specified service subclass.
APP_RQSTS_COMPLETED _TOTAL	BIGINT	Total number of external (application) requests that completed successfully across all members of the database for the specified service subclass
AVG_RQST_CPU_TIME	BIGINT	Average amount of CPU time, in microseconds, used by all external requests that completed successfully. It represents the total of both user and system CPU time.
ROUTINE_TIME_ RQST_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent working on requests that was spent executing user routines.
RQST_WAIT_ TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent working on requests that was spent waiting within the DB2 database server.
ACT_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of the time spent executing activities that was spent waiting within the DB2 database server.

Table 146. Information returned by the MON_WORKLOAD_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
IO_WAIT_TIME_PERCENT	DECIMAL(5,2)	The percentage of the time spent waiting within the DB2 database server that was due to I/O operations. This includes time spent performing direct reads or direct writes, and time spent reading data and index pages from the table space to the bufferpool or writing them back to disk.
LOCK_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent waiting on locks.
AGENT_WAIT_TIME_ PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent by an application queued to wait for an agent under concentrator configurations.
NETWORK_WAIT_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time spent waiting within the DB2 database server that was spent on client-server communications. This includes time spent sending and receiving data over TCP/IP or using the IPC protocol.
SECTION_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent executing sections. This includes the time spent performing sorts.
SECTION_SORT_ PROC_TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing sorts while executing sections.
COMPILE_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent compiling an SQL statement. This includes explicit and implicit compile times.

Table 146. Information returned by the MON_WORKLOAD_SUMMARY administrative view (continued)

Column name	Data type	Description or Monitor element
TRANSACT_END_PROC _TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent performing commit processing or rolling back transactions.
UTILS_PROC_ TIME_PERCENT	DECIMAL(5,2)	The percentage of time the database server spent actively working on requests that was spent running utilities. This includes performing RUNSTATS, reorganization, and load operations.
AVG_LOCK_WAITS _PER_ACT	BIGINT	The average number of times that applications or connections waited for locks per coordinator activities (successful and aborted).
AVG_LOCK_TIMEOUTS _PER_ACT	BIGINT	The average number of times that a request to lock an object timed out per coordinator activities (successful and aborted).
AVG_DEADLOCKS_ PER_ACT	BIGINT	The average number of deadlocks per coordinator activities (successful and aborted).
AVG_LOCK_ESCALS _PER_ACT	BIGINT	The average number of times that locks have been escalated from several row locks to a table lock per coordinator activities (successful and aborted).
ROWS_READ_PER_ ROWS_RETURNED	BIGINT	The average number of rows read from the table per rows returned to the application.
TOTAL_BP_HIT_ RATIO_PERCENT	DECIMAL(5,2)	The percentage of time that the database manager did not need to load a page from disk to service a data or index page request, including requests for XML storage objects (XDAs).

Chapter 13. MQSeries routines

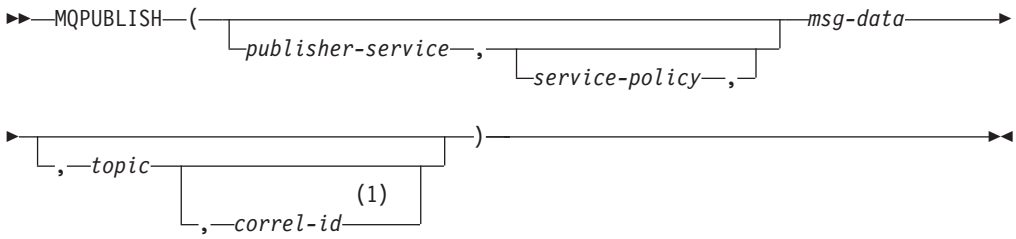
MQPUBLISH

The MQPUBLISH function publishes data to MQSeries. For more details, visit `http://www.ibm.com/software/MQSeries`.

The MQPUBLISH function publishes the data contained in *msg-data* to the MQSeries publisher specified in *publisher-service*, and using the quality of service policy defined by *service-policy*. An optional topic for the message can be specified, and an optional user-defined message correlation identifier can also be specified.

The data type of the result is VARCHAR(1). The result of the function is '1' if successful or '0' if unsuccessful.

Syntax



Notes:

1 The *correl-id* cannot be specified unless a *service* and a *policy* are also specified.

The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Function parameters

publisher-service
A string containing the logical MQSeries destination where the message is to be sent. If specified, the publisher-service must refer to a publisher Service Point defined in the DB2MQ.MQPUBSUB table that has a type value of 'P' for publisher service. If publisher-service is not specified, the DB2.DEFAULT.PUBLISHER will be used. The maximum size of *publisher-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy to be used in handling of this message. If specified, the *service-policy* must refer to a Policy defined in the DB2MQ.MQPOLICY table. A Service Policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

msg-data
A string expression containing the data to be sent via MQSeries. The maximum

size for a VARCHAR string expression is 32 000 bytes and the maximum size for a CLOB string expression is 1M bytes.

topic
A string expression containing the topic for the message publication. If no topic is specified, none will be associated with the message. The maximum size of *topic* is 40 bytes. Multiple topics can be specified in one string (up to 40 characters long). Each topic must be separated by a colon. For example, "t1:t2:the third topic" indicates that the message is associated with all three topics: t1, t2, and "the third topic".

correl-id
An optional string expression containing a correlation identifier to be associated with this message. The *correl-id* is often specified in request and reply scenarios to associate requests with replies. If not specified, no correlation ID will be added to the message. The maximum size of *correl-id* is 24 bytes.

Examples

Example 1: This example publishes the string "Testing 123" to the default publisher service (DB2.DEFAULT.PUBLISHER) using the default policy (DB2.DEFAULT.POLICY). No correlation identifier or topic is specified for the message.

```
VALUES MQPUBLISH('Testing 123')
```

Example 2: This example publishes the string "Testing 345" to the publisher service "MYPUBLISHER" under the topic "TESTS". The default policy is used and no correlation identifier is specified.

```
VALUES MQPUBLISH('MYPUBLISHER','Testing 345', 'TESTS')
```

Example 3: This example publishes the string "Testing 678" to the publisher service "MYPUBLISHER" using the policy "MYPOLICY" with a correlation identifier of "TEST1". The message is published with topic "TESTS".

```
VALUES MQPUBLISH('MYPUBLISHER','MYPOLICY','Testing 678','TESTS','TEST1')
```

Example 4: This example publishes the string "Testing 901" to the publisher service "MYPUBLISHER" under the topic "TESTS" using the default policy (DB2.DEFAULT.POLICY) and no correlation identifier.

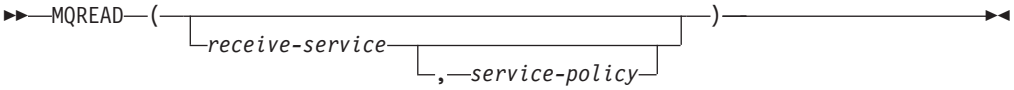
```
VALUES MQPUBLISH('Testing 901','TESTS')
```

MQREAD

The MQREAD function returns a message from the MQSeries location specified by *receive-service*, using the quality of service policy defined in *service-policy*. Executing this operation does not remove the message from the queue associated with *receive-service*, but instead returns the message at the head of the queue.

The data type of the result is VARCHAR (32000). If no messages are available to be returned, the result is the null value.

Syntax



The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Function parameters

receive-service
A string containing the logical MQSeries destination from where the message is to be received. If specified, the *receive-service* must refer to a Service Point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from where a message is sent or received. Service points definitions include the name of the MQSeries Queue Manager and Queue. If *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy used in handling this message. If specified, the *service-policy* must refer to a Policy defined in the DB2MQ.MQPOLICY table. A Service Policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

Examples

Example 1: This example reads the message at the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQREAD()
```

Example 2: This example reads the message at the head of the queue specified by the service "MYSERVICE" using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQREAD('MYSERVICE')
```

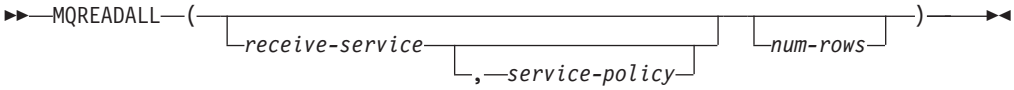
Example 3: This example reads the message at the head of the queue specified by the service "MYSERVICE", and using the policy "MYPOLICY".

```
VALUES MQREAD('MYSERVICE','MYPOLICY')
```

MQREADALL

The MQREADALL table function returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*, using the quality of service policy *service-policy*. Performing this operation does not remove the messages from the queue associated with *receive-service*.

Syntax



The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Table function parameters

receive-service
A string containing the logical MQSeries destination from which the message is read. If specified, the *receive-service* must refer to a service point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from which a message is sent or received. Service point definitions include the name of the MQSeries Queue Manager and Queue. If *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy used in the handling of this message. If specified, the *service-policy* refers to a Policy defined in the DB2MQ.MQPOLICY table. A service policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

num-rows
A positive integer containing the maximum number of messages to be returned by the function.

If *num-rows* is specified, then a maximum of *num-rows* messages will be returned. If *num-rows* is not specified, then all available messages will be returned.

Authorization

EXECUTE privilege on the MQREADALL table function.

Examples

Example 1: This example receives all the messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). The messages and all the metadata are returned as a table.

```
SELECT * FROM table (MQREADALL()) AS T
```

Example 2: This example receives all the messages from the head of the queue specified by the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY). Only the MSG and CORRELID columns are returned.

```
SELECT T.MSG, T.CORRELID FROM table (MQREADALL('MYSERVICE')) AS T
```

Example 3: This example reads the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). Only messages with a CORRELID of '1234' are returned. All columns are returned.

```
SELECT * FROM table (MQREADALL()) AS T WHERE T.CORRELID = '1234'
```

Example 4: This example receives the first 10 messages from the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). All columns are returned.

```
SELECT * FROM table (MQREADALL(10)) AS T
```

Information returned

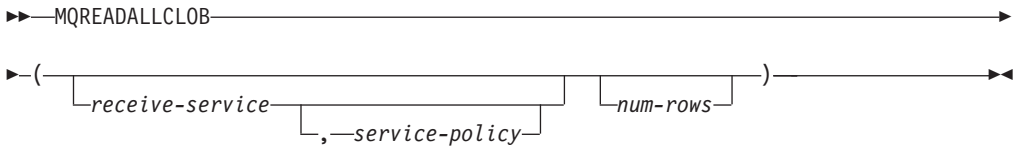
Table 147. Information returned by the MQREADALL table function

Column name	Data type	Description
MSG	VARCHAR(32000)	Contains the contents of the MQSeries message.
CORRELID	VARCHAR(24)	Contains a correlation ID that can be used to identify messages. You can select a message from the queue using this identifier. In the case of a request and response scenario, the correlation ID enables you to associate a response with a particular request.
TOPIC	VARCHAR(40)	Contains the topic with which the message was published, if available.
QNAME	VARCHAR(48)	Contains the name of the queue where the message was received.
MSGID	CHAR(24)	Contains the assigned unique MQSeries identifier for this message.
MSGFORMAT	VARCHAR(8)	Contains the format of the message, as defined by MQSeries. Typical strings have an MQSTR format.

MQREADALLCLOB

The MQREADALLCLOB table function returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*, using the quality of service policy *service-policy*. Performing this operation does not remove the messages from the queue associated with *receive-service*.

Syntax



The schema is DB2MQ.

Table function parameters

receive-service

A string containing the logical MQSeries destination from which the message is read. If specified, the *receive-service* must refer to a service point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from which a message is sent or received. Service point definitions include the name of the

MQSeries Queue Manager and Queue. If *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy used in the handling of this message. If specified, the *service-policy* refers to a Policy defined in the DB2MQ.MQPOLICY table. A service policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

num-rows
A positive integer containing the maximum number of messages to be returned by the function.

If *num-rows* is specified, then a maximum of *num-rows* messages will be returned. If *num-rows* is not specified, then all available messages will be returned.

Authorization

EXECUTE privilege on the MQREADALLCLOB table function.

Examples

Example 1: This example receives all the messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). The messages and all the metadata are returned as a table.

```
SELECT * FROM table (MQREADALLCLOB()) AS T
```

Example 2: This example receives all the messages from the head of the queue specified by the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY). Only the MSG and CORRELID columns are returned.

```
SELECT T.MSG, T.CORRELID FROM table (MQREADALLCLOB('MYSERVICE')) AS T
```

Example 3: This example reads the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). Only messages with a CORRELID of '1234' are returned. All columns are returned.

```
SELECT * FROM table (MQREADALLCLOB()) AS T WHERE T.CORRELID = '1234'
```

Example 4: This example receives the first 10 messages from the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). All columns are returned.

```
SELECT * FROM table (MQREADALLCLOB(10)) AS T
```

Information returned

Table 148. Information returned by the MQREADALLCLOB table function

Column name	Data type	Description
MSG	CLOB(1M)	Contains the contents of the MQSeries message.

Table 148. Information returned by the MQREADALLCLOB table function (continued)

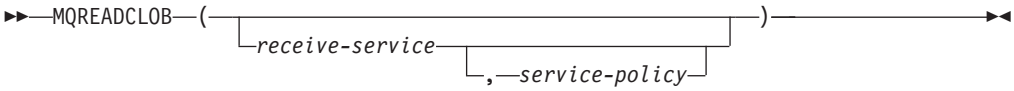
Column name	Data type	Description
CORRELID	VARCHAR(24)	Contains a correlation ID that can be used to identify messages. You can select a message from the queue using this identifier. In the case of a request and response scenario, the correlation ID enables you to associate a response with a particular request.
TOPIC	VARCHAR(40)	Contains the topic with which the message was published, if available.
QNAME	VARCHAR(48)	Contains the name of the queue where the message was received.
MSGID	CHAR(24)	Contains the assigned unique MQSeries identifier for this message.
MSGFORMAT	VARCHAR(8)	Contains the format of the message, as defined by MQSeries. Typical strings have an MQSTR format.

MQREADCLOB

The MQREADCLOB function returns a message from the MQSeries location specified by *receive-service*, using the quality of service policy defined in *service-policy*. Executing this operation does not remove the message from the queue associated with *receive-service*, but instead returns the message at the head of the queue.

The data type of the result is CLOB(1M). If no messages are available to be returned, the result is the null value.

Syntax



The schema is DB2MQ.

Function parameters

receive-service

A string containing the logical MQSeries destination from where the message is to be received. If specified, the *receive-service* must refer to a Service Point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from where a message is sent or received. Service points definitions

include the name of the MQSeries Queue Manager and Queue. If *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy used in handling this message. If specified, the *service-policy* must refer to a Policy defined in the DB2MQ.MQPOLICY table. A Service Policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

Examples

Example 1: This example reads the message at the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQREADCLOB()
```

Example 2: This example reads the message at the head of the queue specified by the service "MYSERVICE" using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQREADCLOB('MYSERVICE')
```

Example 3: This example reads the message at the head of the queue specified by the service "MYSERVICE", and using the policy "MYPOLICY".

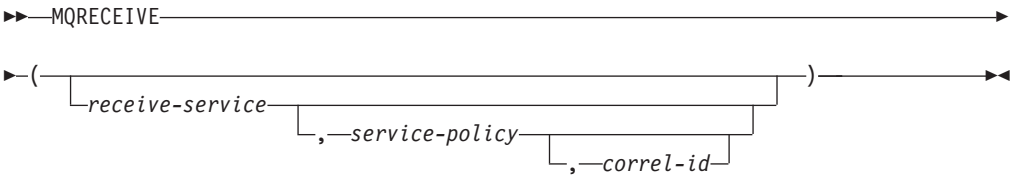
```
VALUES MQREADCLOB('MYSERVICE','MYPOLICY')
```

MQRECEIVE

The MQRECEIVE function returns a message from the MQSeries location specified by *receive-service*, using the quality of service policy *service-policy*. Performing this operation removes the message from the queue associated with *receive-service*. If the *correl-id* is specified, then the first message with a matching correlation identifier will be returned. If *correl-id* is not specified, then the message at the head of the queue will be returned.

The data type of the result is VARCHAR (32000). If no messages are available to be returned, the result is the null value.

Syntax



The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Function parameters

receive-service
A string containing the logical MQSeries destination from which the message is received. If specified, the *receive-service* must refer to a Service Point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from which a message is sent or received. Service points definitions include the name of the MQSeries Queue Manager and Queue. If *receive-service* is not specified, the DB2.DEFAULT.SERVICE is used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy to be used in the handling of this message. If specified, *service-policy* must refer to a policy defined in the DB2MQ.MQPOLICY table. A service policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, the default DB2.DEFAULT.POLICY is used. The maximum size of *service-policy* is 48 bytes.

correl-id
A string containing an optional correlation identifier to be associated with this message. The *correl-id* is often specified in request and reply scenarios to associate requests with replies. If not specified, no correlation id will be specified. The maximum size of *correl-id* is 24 bytes.

Examples

Example 1: This example receives the message at the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQRECEIVE()
```

Example 2: This example receives the message at the head of the queue specified by the service "MYSERVICE" using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQRECEIVE('MYSERVICE')
```

Example 3: This example receives the message at the head of the queue specified by the service "MYSERVICE" using the policy "MYPOLICY".

```
VALUES MQRECEIVE('MYSERVICE','MYPOLICY')
```

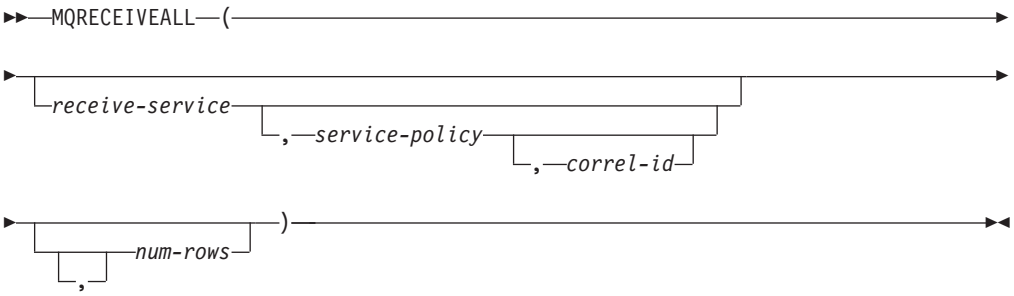
Example 4: This example receives the first message with a correlation id that matches '1234' from the head of the queue specified by the service "MYSERVICE" using the policy "MYPOLICY".

```
VALUES MQRECEIVE('MYSERVICE','MYPOLICY','1234')
```

MQRECEIVEALL

The MQRECEIVEALL table function returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*, using the quality of service policy *service-policy*. Performing this operation removes the messages from the queue associated with *receive-service*.

Syntax



The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Table function parameters

receive-service
A string containing the logical MQSeries destination from which the message is received. If specified, the *receive-service* must refer to a service point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from which a message is sent or received. Service point definitions include the name of the MQSeries Queue Manager and Queue. If *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy used in the handling of this message. If specified, the *service-policy* refers to a Policy defined in the DB2MQ.MQPOLICY table. A service policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

correl-id
An optional string containing a correlation identifier associated with this message. The *correl-id* is often specified in request and reply scenarios to associate requests with replies. If not specified, no correlation id is specified. The maximum size of correl-id is 24 bytes.

If a *correl-id* is specified, all the messages with a matching correlation identifier are returned and removed from the queue. If *correl-id* is not specified, the message at the head of the queue is returned.

num-rows
A positive integer containing the maximum number of messages to be returned by the function.

If *num-rows* is specified, then a maximum of *num-rows* messages will be returned. If *num-rows* is not specified, then all available messages will be returned.

Authorization

EXECUTE privilege on the MQRECEIVEALL table function.

Examples

Example 1: This example receives all the messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). The messages and all the metadata are returned as a table.

```
SELECT * FROM table (MQRECEIVEALL()) AS T
```

Example 2: This example receives all the messages from the head of the queue specified by the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY). Only the MSG and CORRELID columns are returned.

```
SELECT T.MSG, T.CORRELID FROM table (MQRECEIVEALL('MYSERVICE')) AS T
```

Example 3: This example receives all of the message from the head of the queue specified by the service "MYSERVICE", using the policy "MYPOLICY". Only messages with a CORRELID of '1234' are returned. Only the MSG and CORRELID columns are returned.

```
SELECT T.MSG, T.CORRELID FROM table
   (MQRECEIVEALL('MYSERVICE','MYPOLICY','1234')) AS T
```

Example 4: This example receives the first 10 messages from the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). All columns are returned.

```
SELECT * FROM table (MQRECEIVEALL(10)) AS T
```

Information returned

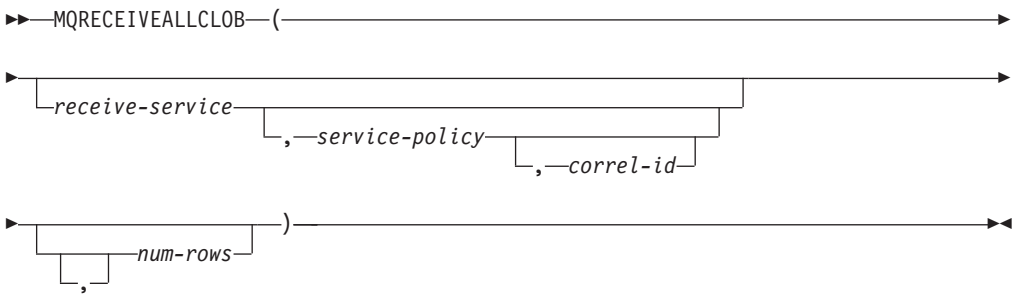
Table 149. Information returned by the MQRECEIVEALL table function

Column name	Data type	Description
MSG	VARCHAR(32000)	Contains the contents of the MQSeries message.
CORRELID	VARCHAR(24)	Contains a correlation ID that can be used to identify messages. You can select a message from the queue using this identifier. In the case of a request and response scenario, the correlation ID enables you to associate a response with a particular request.
TOPIC	VARCHAR(40)	Contains the topic with which the message was published, if available.
QNAME	VARCHAR(48)	Contains the name of the queue where the message was received.
MSGID	CHAR(24)	Contains the assigned unique MQSeries identifier for this message.
MSGFORMAT	VARCHAR(8)	Contains the format of the message, as defined by MQSeries. Typical strings have an MQSTR format.

MQRECEIVEALLCLOB

The MQRECEIVEALLCLOB table function returns a table containing the messages and message metadata from the MQSeries location specified by *receive-service*, using the quality of service policy *service-policy*. Performing this operation removes the messages from the queue associated with *receive-service*.

Syntax



The schema is DB2MQ.

Table function parameters

receive-service
A string containing the logical MQSeries destination from which the message is received. If specified, the *receive-service* must refer to a service point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from which a message is sent or received. Service point definitions include the name of the MQSeries Queue Manager and Queue. If *receive-service* is not specified, then the DB2.DEFAULT.SERVICE will be used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy used in the handling of this message. If specified, the *service-policy* refers to a Policy defined in the DB2MQ.MQPOLICY table A service policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

correl-id
An optional string containing a correlation identifier associated with this message. The *correl-id* is often specified in request and reply scenarios to associate requests with replies. If not specified, no correlation id is specified. The maximum size of correl-id is 24 bytes.

If a *correl-id* is specified, then only those messages with a matching correlation identifier will be returned. If *correl-id* is not specified, then the message at the head of the queue will be returned.

num-rows
A positive integer containing the maximum number of messages to be returned by the function.

If *num-rows* is specified, then a maximum of *num-rows* messages will be returned. If *num-rows* is not specified, then all available messages are returned.

Authorization

EXECUTE privilege on the MQRECEIVEALLCLOB table function.

Examples

Example 1: This example receives all the messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). The messages and all the metadata are returned as a table.

```
SELECT * FROM table (MQRECEIVEALLCLOB()) AS T
```

Example 2: This example receives all the messages from the head of the queue specified by the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY). Only the MSG and CORRELID columns are returned.

```
SELECT T.MSG, T.CORRELID
   FROM table (MQRECEIVEALLCLOB('MYSERVICE')) AS T
```

Example 3: This example receives all of the message from the head of the queue specified by the service "MYSERVICE", using the policy "MYPOLICY". Only messages with a CORRELID of '1234' are returned. Only the MSG and CORRELID columns are returned.

```
SELECT T.MSG, T.CORRELID
   FROM table (MQRECEIVEALLCLOB('MYSERVICE','MYPOLICY','1234')) AS T
```

Example 4: This example receives the first 10 messages from the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). All columns are returned.

```
SELECT * FROM table (MQRECEIVEALLCLOB(10)) AS T
```

Information returned

Table 150. Information returned by the MQRECEIVEALLCLOB table function

Column name	Data type	Description
MSG	CLOB(1M)	Contains the contents of the MQSeries message.
CORRELID	VARCHAR(24)	Contains a correlation ID that can be used to identify messages. You can select a message from the queue using this identifier. In the case of a request and response scenario, the correlation ID enables you to associate a response with a particular request.
TOPIC	VARCHAR(40)	Contains the topic with which the message was published, if available.
QNAME	VARCHAR(48)	Contains the name of the queue where the message was received.
MSGID	CHAR(24)	Contains the assigned unique MQSeries identifier for this message.

Table 150. Information returned by the MQRECEIVEALLCLOB table function (continued)

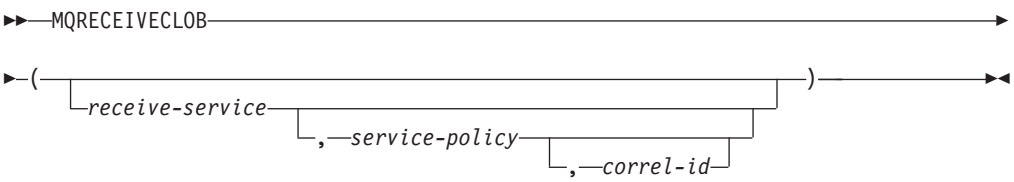
Column name	Data type	Description
MSGFORMAT	VARCHAR(8)	Contains the format of the message, as defined by MQSeries. Typical strings have an MQSTR format.

MQRECEIVECLOB

The MQRECEIVECLOB function returns a message from the MQSeries location specified by *receive-service*, using the quality of service policy *service-policy*. Performing this operation removes the message from the queue associated with *receive-service*. If the *correl-id* is specified, the first message with a matching correlation identifier will be returned. If *correl-id* is not specified, the message at the head of the queue will be returned.

The data type of the result is CLOB(1M). If no messages are available to be returned, the result is the null value.

Syntax



The schema is DB2MQ.

Function parameters

receive-service
A string containing the logical MQSeries destination from which the message is received. If specified, the *receive-service* must refer to a Service Point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from which a message is sent or received. Service points definitions include the name of the MQSeries Queue Manager and Queue. If *receive-service* is not specified, the DB2.DEFAULT.SERVICE is used. The maximum size of *receive-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy to be used in the handling of this message. If specified, the *service-policy* must refer to a policy defined in the DB2MQ.MQPOLICY table. A service policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, the default DB2.DEFAULT.POLICY is used. The maximum size of *service-policy* is 48 bytes.

correl-id
A string containing an optional correlation identifier to be associated with this message. The *correl-id* is often specified in request and reply scenarios to associate requests with replies. If not specified, no correlation id will be used. The maximum size of *correl-id* is 24 bytes.

Examples

Example 1: This example receives the message at the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQRECEIVECLOB()
```

Example 2: This example receives the message at the head of the queue specified by the service "MYSERVICE" using the default policy (DB2.DEFAULT.POLICY).

```
VALUES MQRECEIVECLOB('MYSERVICE')
```

Example 3: This example receives the message at the head of the queue specified by the service "MYSERVICE" using the policy "MYPOLICY".

```
VALUES MQRECEIVECLOB('MYSERVICE','MYPOLICY')
```

Example 4: This example receives the first message with a correlation ID that matches '1234' from the head of the queue specified by the service "MYSERVICE" using the policy "MYPOLICY".

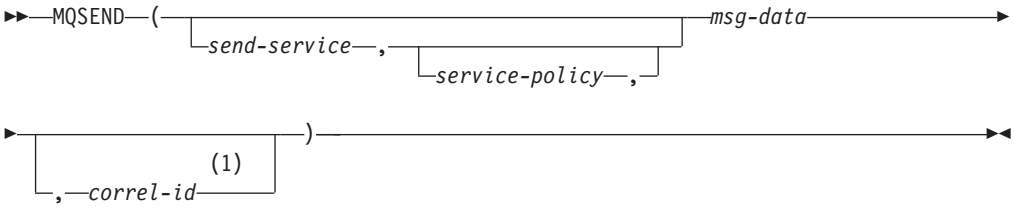
```
VALUES MQRECEIVECLOB('MYSERVICE',MYPOLICY','1234')
```

MQSEND

The MQSEND function sends the data contained in *msg-data* to the MQSeries location specified by *send-service*, using the quality of service policy defined by *service-policy*. An optional user-defined message correlation identifier can be specified using *correl-id*.

The data type of the result is VARCHAR(1). The result of the function is '1' if successful or '0' if unsuccessful.

Syntax



Notes:

1 The *correl-id* cannot be specified unless a *service* and a *policy* are also specified.

The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Function parameters

msg-data
A string expression containing the data to be sent via MQSeries. The maximum size for a VARCHAR string expression is 32 000 bytes and the maximum size for a CLOB string expression is 1M bytes.

send-service
A string containing the logical MQSeries destination where the message is to

be sent. If specified, the *send-service* refers to a service point defined in the DB2MQ.MQSERVICE table. A service point is a logical end-point from which a message may be sent or received. Service point definitions include the name of the MQSeries Queue Manager and Queue. If *send-service* is not specified, the value of DB2.DEFAULT.SERVICE is used. The maximum size of *send-service* is 48 bytes.

service-policy
A string containing the MQSeries Service Policy used in handling of this message. If specified, the *service-policy* must refer to a service policy defined in the DB2MQ.MQPOLICY table. A Service Policy defines a set of quality of service options that should be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, a default value of DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

correl-id
An optional string containing a correlation identifier associated with this message. The *correl-id* is often specified in request and reply scenarios to associate requests with replies. If not specified, no correlation ID will be sent. The maximum size of *correl-id* is 24 bytes.

Examples

Example 1: This example sends the string "Testing 123" to the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY), with no correlation identifier.

```
VALUES MQSEND('Testing 123')
```

Example 2: This example sends the string "Testing 345" to the service "MYSERVICE", using the policy "MYPOLICY", with no correlation identifier.

```
VALUES MQSEND('MYSERVICE','MYPOLICY','Testing 345')
```

Example 3: This example sends the string "Testing 678" to the service "MYSERVICE", using the policy "MYPOLICY", with correlation identifier "TEST3".

```
VALUES MQSEND('MYSERVICE','MYPOLICY','Testing 678','TEST3')
```

Example 4: This example sends the string "Testing 901" to the service "MYSERVICE", using the default policy (DB2.DEFAULT.POLICY), and no correlation identifier.

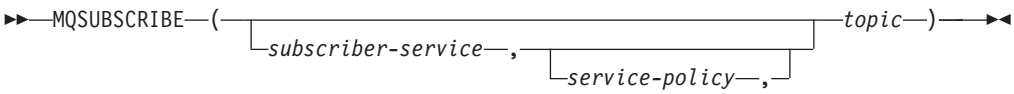
```
VALUES MQSEND('MYSERVICE','Testing 901')
```

MQSUBSCRIBE

The MQSUBSCRIBE function is used to register interest in MQSeries messages published on a specified topic. Successful execution of this function causes the publish and subscribe server to forward messages matching the topic to the service point defined by *subscriber-service*. The *subscriber-service* specifies a logical destination for messages that match the specified topic. Messages that match *topic* are placed on the queue defined by *subscriber-service*, and can be read or received through a subsequent call to MQREAD, MQRECEIVE, MQREADALL, or MQRECEIVEALL. For more details, visit `http://www.ibm.com/software/MQSeries`.

The data type of the result is VARCHAR(1). The result of the function is '1' if successful or '0' if unsuccessful.

Syntax



The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Function parameters

subscriber-service
A string containing the logical MQSeries subscription point to where messages matching *topic* will be sent. If specified, the *subscriber-service* must refer to a Subscribers Service Point defined in the DB2MQ.MQPUBSUB table that has a type value of 'S' for publisher service. If *subscriber-service* is not specified, then the DB2.DEFAULT.SUBSCRIBER will be used instead. The maximum size of subscriber-service is 48 bytes.

service-policy
A string containing the MQSeries Service Policy to be used in handling the message. If specified, the *service-policy* must refer to a Policy defined in the DB2MQ.MQPOLICY table. A Service Policy defines a set of quality of service options to be applied to this messaging operation. These options include message priority and message persistence. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used instead. The maximum size of service-policy is 48 bytes.

topic
A string defining the types of messages to receive. Only messages published with the specified topics will be received by this subscription. Multiple subscriptions can coexist. The maximum size of topic is 40 bytes. Multiple topics can be specified in one string (up to 40 bytes long). Each topic must be separated by a colon. For example, "t1:t2:the third topic" indicates that the message is associated with all three topics: t1, t2, and "the third topic".

Examples

Example 1: This example registers an interest in messages containing the topic "Weather". The default subscriber-service (DB2.DEFAULT.SUBSCRIBER) is registered as the subscriber and the default service-policy (DB2.DEFAULT.POLICY) specifies the quality of service.

```
VALUES MQSUBSCRIBE('Weather')
```

Example 2: This example demonstrates a subscriber registering interest in messages containing "Stocks". The subscriber registers as "PORTFOLIO-UPDATES" with policy "BASIC-POLICY".

```
VALUES MQSUBSCRIBE('PORTFOLIO-UPDATES','BASIC-POLICY','Stocks')
```

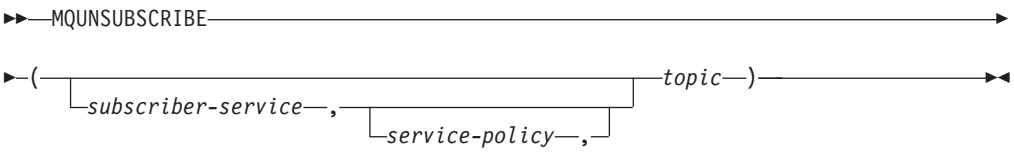
MQUNSUBSCRIBE

The MQUNSUBSCRIBE function is used to unregister an existing message subscription. The *subscriber-service*, *service-policy*, and *topic* are used to identify the subscription that is to be cancelled. Successful execution of this function causes the publish and subscribe server to remove the specified subscription. Messages with

the specified *topic* will no longer be sent to the logical destination defined by *subscriber-service*. For more details, visit `http://www.ibm.com/software/MQSeries`.

The data type of the result is VARCHAR(1). The result of the function is '1' if successful or '0' if unsuccessful.

Syntax



The schema is DB2MQ for non-transactional message queuing functions, and DB2MQ1C for one-phase commit transactional MQ functions.

Function parameters

subscriber-service
If specified, the *subscriber-service* must refer to a Subscribers Service Point defined in the DB2MQ.MQPUBSUB table that has a type value of 'S' for publisher service. If *subscriber-service* is not specified, then the DB2.DEFAULT.SUBSCRIBER will be used instead. The maximum size of subscriber-service is 48 bytes.

service-policy
If specified, the *service-policy* must refer to a Policy defined in the DB2MQ.MQPOLICY table. A Service Policy defines a set of quality of service options to be applied to this messaging operation. If *service-policy* is not specified, then the default DB2.DEFAULT.POLICY will be used. The maximum size of *service-policy* is 48 bytes.

topic
A string specifying the subject of messages that are not to be received. The maximum size of *topic* is 40 bytes. Multiple topics can be specified in one string (up to 40 bytes long). Each topic must be separated by a colon. For example, "t1:t2:the third topic" indicates that the message is associated with all three topics: t1, t2, and "the third topic".

Examples

Example 1: This example cancels an interest in messages containing the topic "Weather". The default subscriber-service (DB2.DEFAULT.SUBSCRIBER) is registered as the unsubscriber and the default service-policy (DB2.DEFAULT.POLICY) specifies the quality of service.

```
VALUES MQUNSUBSCRIBE('Weather')
```

Example 2: This example demonstrates a subscriber canceling an interest in messages containing "Stocks". The subscriber is registered as "PORTFOLIO-UPDATES" with policy "BASIC-POLICY".

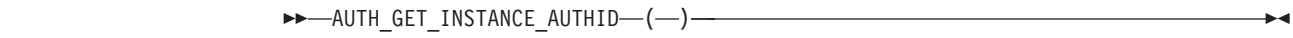
```
VALUES MQUNSUBSCRIBE('PORTFOLIO-UPDATES','BASIC-POLICY','Stocks')
```

Chapter 14. Security routines and views

AUTH_GET_INSTANCE_AUTHID - Get the instance owner authorization ID

The AUTH_GET_INSTANCE_AUTHID scalar function returns the authorization ID of the instance owner

Syntax



The schema is SYSPROC.

Authorization

EXECUTE privilege on the AUTH_GET_INSTANCE_AUTHID scalar function.

Example

The following example shows how to use the DB2 command line processor (CLP) to obtain the authorization ID of the instance owner:

```
db2 "values SYSPROC.AUTH_GET_INSTANCE_AUTHID()"
```

The following is an example of output for this command.

```
1
-------------------------...-------------
ZURBIE

  1 record(s) selected.
```

Usage notes

Common configurations have the instance owner account as a member of the SYSADM group, therefore, prior to DB2 Version 9.7, applications that are run under the instance owner account had unlimited authority on the database. In DB2 Version 9.7, a user who holds SYSADM authority no longer has implicit DBADM authority and as a result applications that are run under the instance owner account might experience authorization errors, such as SQL1092N, SQL0551N, and SQL0552N, when performing operations that are no longer within the scope of SYSADM authority.

The UPGRADE DATABASE command and the RESTORE DATABASE command (for a downlevel database) grant DBADM authority to the SYSADM group, however this is not the case for any new Version 9.7 database.

To obtain a list of the authorities held by the instance owner authorization ID, follow these steps:

1. Use the SYSPROC.AUTH_GET_INSTANCE_AUTHID() scalar function to determine the instance owner authorization ID. For example:

   ```
   db2 "VALUES SYSPROC.AUTH_GET_INSTANCE_AUTHID()"
   ```

This command returns.

```
1
-----------------------------------
BOB

  1 record(s) selected.
```

2. Get a list of the authorities for this authorization ID. For example:

```
SELECT * FROM
   TABLE (SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID ('BOB', 'U') ) AS T
   ORDER BY AUTHORITY
```

3. If necessary, grant any missing authorities. For example:

```
GRANT DBADM ON DATABASE TO USER BOB
```

Information returned

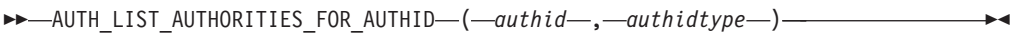
Table 151. Information returned by the AUTH_GET_INSTANCE_AUTHID scalar function

Column name	Data type	Description
InstanceAuthId	VARCHAR(128)	The authorization ID of the instance owner.

AUTH_LIST_AUTHORITIES_FOR_AUTHID

The AUTH_LIST_AUTHORITIES_FOR_AUTHID table function returns all authorities held by the authorization ID, either found in the database configuration file or granted to an authorization ID directly or indirectly through a group or a role.

Syntax



The schema is SYSPROC.

Table function parameters

authid
An input argument of type VARCHAR(128) that specifies the authorization ID being queried. The authorization ID can be a user, group or a role. If *authid* is NULL or an empty string, an empty result table is returned.

authidtype
An input argument of type VARCHAR(1) that specifies the authorization ID type being queried. If *authidtype* does not exist, is NULL or an empty string, an empty result table is returned. Possible values for *authidtype* are:

- G: Group
- R: Role
- U: User

Authorization

EXECUTE privilege on the AUTH_LIST_AUTHORITIES_FOR_AUTHID function.

Information returned

Table 152. The information returned for AUTH_LIST_AUTHORITIES_FOR_AUTHID

Column Name	Data Type	Description
AUTHORITY	VARCHAR(128)	Authority held by the authorization ID
D_USER	CHAR(1)	Authority granted directly to the *authid*, when the *authidtype* is a user (U). If the *authidtype* is a group (G) or a role (R), then the value is not applicable ('*'). • N = Not held • Y= Held • * = Not applicable
D_GROUP	CHAR(1)	Authority granted directly to the *authid* when the *authidtype* is a group (G), or to the group to which the *authid* belongs when the *authidtype* is a user (U). If the *authidtype* is a role (R), then the value is not applicable ('*'). • N = Not held • Y= Held • * = Not applicable
D_PUBLIC	CHAR(1)	Authority granted directly to the *authid* called PUBLIC when the *authidtype* is a user (U) or a group (G). If the *authidtype* is a role (R), then the value is not applicable ('*'). • N = Not held • Y= Held • * = Not applicable
ROLE_USER	CHAR(1)	Authority granted directly to a role granted the *authid*, when the *authidtype* is a user (U). If the *authidtype* is a group (G) or a role (R), then the value is not applicable ('*'). The role could be part of a role hierarchy. • N = Not held • Y= Held • * = Not applicable
ROLE_GROUP	CHAR(1)	Authority granted directly to a role granted to the *authid* when the *authidtype* is a group (G). If the *authidtype* is a user (U) or a role (R), then the value is not applicable ('*'). The role could be part of a role hierarchy. • N = Not held • Y= Held • * = Not applicable
ROLE_PUBLIC	CHAR(1)	Authority granted directly to a role granted to the *authid* called PUBLIC when the *authidtype* is a user (U) or a group (G). If the *authidtype* is a role (R), then the value is not applicable ('*'). The role could be part of a role hierarchy. • N = Not held • Y= Held • * = Not applicable
D_ROLE	CHAR(1)	Authority granted to a role or to a role granted to the role. If the *authidtype* is a user (U) or a group (G), then the value is not applicable ('*'). The role could be part of a role hierarchy. • N = Not held • Y= Held • * = Not applicable

Example

Consider user ALICE who by default holds BIND, CONNECT, CREATETAB and IMPLICIT_SCHEMA privileges through special group PUBLIC. ALICE is a member of a group ADMIN1 who has the following system authorities: SYSADM, SYSCTRL and SYSMAINT. She is also a member of group ADMIN2 who has DBADM authority. Also, ALICE has been granted DBADM and SECADM database authorities. Role R1 was granted to ALICE. LOAD authority was granted to role R1. Role R2 was granted to group ADMIN1. CREATE_NOT_FENCED_ROUTINE authority was granted to role R2.

Example 1: Retrieve all authorities user ALICE has granted either directly to her or indirectly through a group, PUBLIC or a role.

```
SELECT AUTHORITY, D_USER, D_GROUP, D_PUBLIC, ROLE_USER, ROLE_GROUP, ROLE_PUBLIC, D_ROLE
   FROM TABLE (SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID ('ALICE', 'U') ) AS T
   ORDER BY AUTHORITY
```

```
AUTHORITY                D_USER D_GROUP D_PUBLIC ROLE_USER ROLE_GROUP ROLE_PUBLIC D_ROLE
------------------------ ------ ------- -------- --------- ---------- ----------- ------
ACCESSCTRL               N      N       N        N         N          N           *
BINDADD                  N      N       Y        N         N          N           *
CONNECT                  N      N       Y        N         N          N           *
CREATE_EXTERNAL_ROUTINE  N      N       N        N         N          N           *
CREATE_NOT_FENCED_ROUTINE N     N       N        N         Y          N           *
CREATETAB                N      N       Y        N         N          N           *
DATAACCESS               N      N       N        N         N          N           *
DBADM                    Y      Y       N        N         N          N           *
EXPLAIN                  N      N       N        N         N          N           *
IMPLICIT_SCHEMA          N      N       Y        N         N          N           *
LOAD                     N      N       N        Y         N          N           *
QUIESCE_CONNECT          N      N       N        N         N          N           *
SECADM                   Y      N       N        N         N          N           *
SQLADM                   N      N       N        N         N          N           *
SYSADM                   *      Y       *        *         *          *           *
SYSCTRL                  *      Y       *        *         *          *           *
SYSMAINT                 *      Y       *        *         *          *           *
SYSMON                   *      N       *        *         *          *           *
WLMADM                   N      N       N        N         N          N           *
```

Example 2: Retrieve all authorities group ADMIN1 has granted either directly to it or indirectly through PUBLIC or a role.

```
SELECT AUTHORITY, D_USER, D_GROUP, D_PUBLIC, ROLE_USER, ROLE_GROUP, ROLE_PUBLIC, D_ROLE
   FROM TABLE (SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID ('ADMIN1', 'G') ) AS T
   ORDER BY AUTHORITY
```

```
AUTHORITY                D_USER D_GROUP D_PUBLIC ROLE_USER ROLE_GROUP ROLE_PUBLIC D_ROLE
------------------------ ------ ------- -------- --------- ---------- ----------- ------
ACCESSCTRL               *      N       *        *         N          *           *
BINDADD                  *      N       *        *         N          *           *
CONNECT                  *      N       *        *         N          *           *
CREATE_EXTERNAL_ROUTINE  *      N       *        *         N          *           *
CREATE_NOT_FENCED_ROUTINE *     N       *        *         Y          *           *
CREATETAB                *      N       *        *         N          *           *
DATAACCESS               *      N       *        *         N          *           *
DBADM                    *      N       *        *         N          *           *
EXPLAIN                  *      N       *        *         N          *           *
IMPLICIT_SCHEMA          *      N       *        *         N          *           *
LOAD                     *      N       *        *         N          *           *
QUIESCE_CONNECT          *      N       *        *         N          *           *
SECADM                   *      N       *        *         N          *           *
SQLADM                   *      N       *        *         N          *           *
SYSADM                   *      Y       *        *         *          *           *
SYSCTRL                  *      Y       *        *         *          *           *
SYSMAINT                 *      Y       *        *         *          *           *
SYSMON                   *      N       *        *         *          *           *
WLMADM                   *      N       *        *         N          *           *
```

Example 3: Retrieve all authorities special group PUBLIC has granted either directly to it or indirectly through a role

```
SELECT AUTHORITY, D_USER, D_GROUP, D_PUBLIC, ROLE_USER, ROLE_GROUP, ROLE_PUBLIC, D_ROLE
   FROM TABLE (SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID ('PUBLIC', 'G') ) AS T
   ORDER BY AUTHORITY

1                        D_USER D_GROUP D_PUBLIC ROLE_USER ROLE_GROUP ROLE_PUBLIC D_ROLE
------------------------ ------ ------- -------- --------- ---------- ----------- ------
ACCESSCTRL               *      *       N        *         *          N           *
BINDADD                  *      *       Y        *         *          N           *
CONNECT                  *      *       Y        *         *          N           *
CREATE_EXTERNAL_ROUTINE  *      *       N        *         *          N           *
CREATE_NOT_FENCED_ROUTINE *     *       N        *         *          N           *
CREATETAB                *      *       Y        *         *          N           *
DATAACCESS               *      *       N        *         *          N           *
DBADM                    *      *       N        *         *          N           *
EXPLAIN                  *      *       N        *         *          N           *
IMPLICIT_SCHEMA          *      *       Y        *         *          N           *
LOAD                     *      *       N        *         *          N           *
QUIESCE_CONNECT          *      *       N        *         *          N           *
SECADM                   *      *       N        *         *          N           *
SQLADM                   *      *       N        *         *          N           *
SYSADM                   *      *       *        *         *          *           *
SYSCTRL                  *      *       *        *         *          *           *
SYSMAINT                 *      *       *        *         *          *           *
SYSMON                   *      *       *        *         *          *           *
WLMADM                   *      *       N        *         *          N           *
```

Example 4: Retrieve all authorities role R1 has granted either directly to it or indirectly through a role. Consider in this case that role R2 was also granted to role R1.

```
SELECT AUTHORITY, D_USER, D_GROUP, D_PUBLIC, ROLE_USER, ROLE_GROUP, ROLE_PUBLIC, D_ROLE
   FROM TABLE (SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID ('R1', 'R') ) AS T
   ORDER BY AUTHORITY

AUTHORITY                D_USER D_GROUP D_PUBLIC ROLE_USER ROLE_GROUP ROLE_PUBLIC D_ROLE
------------------------ ------ ------- -------- --------- ---------- ----------- ------
ACCESSCTRL               *      *       *        *         *          *           N
BINDADD                  *      *       *        *         *          *           N
CONNECT                  *      *       *        *         *          *           N
CREATE_EXTERNAL_ROUTINE  *      *       *        *         *          *           N
CREATE_NOT_FENCED_ROUTINE *     *       *        *         *          *           Y
CREATETAB                *      *       *        *         *          *           N
DATAACCESS               *      *       *        *         *          *           N
DBADM                    *      *       *        *         *          *           N
EXPLAIN                  *      *       *        *         *          *           N
IMPLICIT_SCHEMA          *      *       *        *         *          *           N
LOAD                     *      *       *        *         *          *           Y
QUIESCE_CONNECT          *      *       *        *         *          *           N
SECADM                   *      *       *        *         *          *           N
SYSADM                   *      *       *        *         *          *           *
SQLADM                   *      *       *        *         *          *           N
SYSCTRL                  *      *       *        *         *          *           *
SYSMAINT                 *      *       *        *         *          *           *
SYSMON                   *      *       *        *         *          *           *
WLMADM                   *      *       *        *         *          *           N
```

Usage Notes

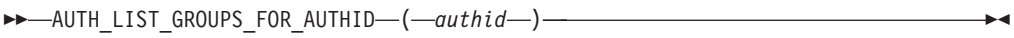
The output of AUTH_LIST_AUTHORITIES_FOR_AUTHID table function depends on the *authidtype*. For example, for an *authidtype* of USER, it returns all authorities that *authid* holds through any means:

- granted directly to the *authid*
- granted to any group (or roles granted to the group) to which *authid* belongs
- granted to any role (or roles granted to the role) granted to *authid*
- granted to PUBLIC (or roles granted to PUBLIC)

AUTH_LIST_GROUPS_FOR_AUTHID table function - Retrieve group membership list for a given authorization ID

The AUTH_LIST_GROUPS_FOR_AUTHID table function returns the list of groups of which the given authorization ID is a member.

Syntax



The schema is SYSPROC.

Table function parameter

authid
An input argument of type VARCHAR(128) that specifies the authorization ID being queried. The authorization ID can only represent a user. If *authid* does not exist, is NULL or empty string, an empty result table is returned.

Authorization

EXECUTE privilege on the AUTH_LIST_GROUPS_FOR_AUTHID table function.

Example

Retrieve all groups that AMY belongs to.

```
SELECT * FROM TABLE (SYSPROC.AUTH_LIST_GROUPS_FOR_AUTHID('AMY')) AS T
```

The following is an example of output for this query.

```
GROUP
-------------------------...-------------
BUILD
PDXDB2

  2 record(s) selected.
```

Usage notes

Group information returned might be different than expected for the following reasons:

- In a Windows Active Directory environment, the database manager:
  - supports one level of group nesting within a local group, except the nesting of a domain local group within a local group. For example, if *authid* belongs to the global group G1, and G1 belongs to the local group L1, the local group L1 is returned as the group for *authid*. However, if *authid* belongs to the domain local group DL1, and DL1 belongs to the local group L1, no group information is returned for *authid*.
  - does not support any nesting of global groups. For example, if *authid* belongs to the global G2, and G2 belongs to the global G3, only G2 is returned as the group for *authid*.
- The registry variable DB2_GRP_LOOKUP specifies which Windows security mechanism is used to enumerate the groups to which a user belongs.

- For an authorization ID that belongs to a particular domain, if the domain is not specified as part of the *authid*, and both a local and domain *authid* exist with the same name, the groups for the local authorization ID is returned.

Information returned

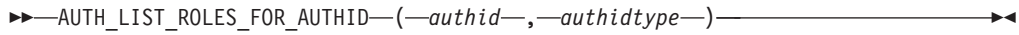
Table 153. Information returned by the AUTH_LIST_GROUPS_FOR_AUTHID table function

Column name	Data type	Description
GROUP	VARCHAR(128)	The group to which the authorization ID belongs.

AUTH_LIST_ROLES_FOR_AUTHID function - Returns the list of roles

The AUTH_LIST_ROLES_FOR_AUTHID function returns the list of roles in which the given authorization ID is a member.

Syntax



The schema is SYSPROC.

Table function parameters

authid
An input argument of type VARCHAR(128) that specifies the authorization ID being queried. The authorization ID can be a user, group or a role. If *authid* is NULL or an empty string, an empty result table is returned.

authidtype
An input argument of type VARCHAR(1) that specifies the authorization ID type being queried. If *authidtype* does not exist, is NULL or an empty string, an empty result table is returned. Possible values for *authidtype* are:

- G: Group
- R: Role
- U: User

Authorization

EXECUTE privilege on the AUTH_LIST_ROLES_FOR_AUTHID function.

Information returned

Table 154. The result sets for AUTH_LIST_ROLES_FOR_AUTHID

Column Name	Data Type	Description
GRANTOR	VARCHAR(128)	Grantor of the role.
GRANTORTYPE	CHAR(1)	Type of grantor: • U = Grantor is an individual user
GRANTEE	VARCHAR(128)	User granted the role.

Table 154. The result sets for AUTH_LIST_ROLES_FOR_AUTHID (continued)

Column Name	Data Type	Description
GRANTEETYPE	CHAR(1)	Type of grantee: • G = Grantee is a group • R= Grantee is a role • U= Grantee is a user
ROLENAME	VARCHAR(128)	Name of the role granted to the authorization ID directly or indirectly through a group or another role.
CREATE_TIME	TIMESTAMP	Time when role was created.
ADMIN	CHAR(1)	Privilege to grant the role to, revoke the role from, or to comment on a role: • N = Not held • Y= Held

Example

Consider granting role INTERN to role DOCTOR and role DOCTOR to role SPECIALIST, then grant role SPECIALIST to user ALICE. ALICE belongs to group HOSPITAL and role EMPLOYEE is granted to group HOSPITAL. ALICE also belongs to special group PUBLIC and role PATIENTS is granted to PUBLIC.

Example 1: Retrieve all roles granted to user ALICE.

```
SELECT GRANTOR, GRANTORTYPE, GRANTEE, GRANTEETYPE, ROLENAME,
  CREATE_TIME, ADMIN
FROM TABLE (SYSPROC.AUTH_LIST_ROLES_FOR_AUTHID ('ALICE', 'U') ) AS T
```

The following is an example of output for this query.

```
GRANTOR GRANTORTYPE GRANTEE    GRANTEETYPE ROLENAME   CREATE_TIME                 ADMIN
------- ----------- -------    ----------- --------   --------------------------- -----
ZURBIE  U           DOCTOR     R           INTERN     2006-08-01-15.09.58.537399 N
ZURBIE  U           SPECIALIST R           DOCTOR     2006-08-01-15.10.04.540660 N
ZURBIE  U           ALICE      U           SPECIALIST 2006-08-01-15.10.08.776218 N
ZURBIE  U           HOSPITAL   G           EMPLOYEE   2006-08-01-15.10.14.277576 N
ZURBIE  U           PUBLIC     G           PATIENTS   2006-08-01-15.10.18.878609 N

  5 record(s) selected.
```

Example 2: Retrieve all roles granted to group HOSPITAL.

```
SELECT GRANTOR, GRANTORTYPE, GRANTEE, GRANTEETYPE, ROLENAME,
  CREATE_TIME, ADMIN
FROM TABLE (SYSPROC.AUTH_LIST_ROLES_FOR_AUTHID ('HOSPITAL', 'G') ) AS T
```

The following is an example of output for this query.

```
GRANTOR GRANTORTYPE GRANTEE    GRANTEETYPE ROLENAME   CREATE_TIME                 ADMIN
------- ----------- -------    ----------- --------   --------------------------- -----
ZURBIE  U           HOSPITAL   G           EMPLOYEE   2006-08-01-15.10.14.277576 N

  1  record(s) selected.
```

Example 3: Retrieve all roles granted to role SPECIALIST.

```
SELECT GRANTOR, GRANTORTYPE, GRANTEE, GRANTEETYPE, ROLENAME,
  CREATE_TIME, ADMIN
FROM TABLE (SYSPROC.AUTH_LIST_ROLES_FOR_AUTHID ('SPECIALIST', 'R') ) AS T
```

The following is an example of output for this query.

```
GRANTOR GRANTORTYPE GRANTEE    GRANTEETYPE ROLENAME   CREATE_TIME               ADMIN
------- ----------- ---------- ----------- ---------- -------------------------- -----
ZURBIE  U           DOCTOR     R           INTERN     2006-08-01-15.09.58.537399 N
ZURBIE  U           SPECIALIST R           DOCTOR     2006-08-01-15.10.04.540660 N

  2  record(s) selected.
```

Example 4: Retrieve all roles granted to group PUBLIC

```
SELECT GRANTOR, GRANTORTYPE, GRANTEE, GRANTEETYPE, ROLENAME,
  CREATE_TIME, ADMIN
FROM TABLE (SYSPROC.AUTH_LIST_ROLES_FOR_AUTHID ('PUBLIC', 'G') ) AS T
```

The following is an example of output for this query.

```
GRANTOR GRANTORTYPE GRANTEE    GRANTEETYPE ROLENAME   CREATE_TIME               ADMIN
------- ----------- ---------- ----------- ---------- -------------------------- -----
ZURBIE  U           PUBLIC     G           PATIENTS   2006-08-01-15.10.18.878609 N

  1 record(s) selected.
```

Usage notes

The output of AUTH_LIST_ROLES_FOR_AUTHID table function depends on the AUTHIDTYPE:

- For a user it returns the roles granted to the user directly or indirectly through another roles, groups that the user belongs to (or PUBLIC).
- For a group it returns the roles granted to the group, directly or indirectly through another roles.
- For a role it returns the roles granted to the role, directly or indirectly through another roles.

AUTHORIZATIONIDS administrative view - Retrieve authorization IDs and types

The AUTHORIZATIONIDS administrative view returns a list of authorization IDs that have been granted privileges or authorities, along with their types, for all authorization IDs defined in the system catalogs from the currently connected database. If privileges or authorities have been granted to groups or roles, only the group or role names are returned.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the AUTHORIZATIONIDS administrative view
- CONTROL privilege on the AUTHORIZATIONIDS administrative view
- DATAACCESS authority

Example

Retrieve all authorization IDs that have been granted privileges or authorities, along with their types.

```
SELECT * FROM SYSIBMADM.AUTHORIZATIONIDS
```

The following is an example of output for this query.

```
AUTHID                          AUTHIDTYPE
------------------------...------- ----------
PUBLIC                          G
JESSICAE                        U
DOCTOR                          R

  3 record(s) selected.
```

Information returned

Table 155. Information returned by the AUTHORIZATIONIDS administrative view

Column name	Data type	Description
AUTHID	VARCHAR(128)	Authorization ID that has been explicitly granted privileges or authorities.
AUTHIDTYPE	CHAR(1)	Authorization ID type: • U: user • R: role • G: group

OBJECTOWNERS administrative view – Retrieve object ownership information

The OBJECTOWNERS administrative view returns all object ownership information for every authorization ID of type USER that owns an object and that is defined in the system catalogs from the currently connected database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the OBJECTOWNERS administrative view
- CONTROL privilege on the OBJECTOWNERS administrative view
- DATAACCESS authority

Example

Retrieve all object ownership information for object schema 'THERESAX'.

```
SELECT SUBSTR(OWNER,1,10) AS OWNER, OWNERTYPE,
   SUBSTR(OBJECTNAME,1,30) AS OBJECTNAME,
   SUBSTR(OBJECTSCHEMA,1,10) AS OBJECTSCHEMA, OBJECTTYPE
   FROM SYSIBMADM.OBJECTOWNERS WHERE OBJECTSCHEMA='THERESAX'
```

The following is an example of output for this query.

```
OWNER      OWNERTYPE OBJECTNAME           OBJECTSCHEMA OBJECTTYPE
---------- --------- -----------------... ------------ ----------------
THERESAX   U         MIN_SALARY           THERESAX     TRIGGER
THERESAX   U         POLICY_IR            SYSTOOLS     TRIGGER
THERESAX   U         CUSTOMER             THERESAX     XML SCHEMA
THERESAX   U         DB2DETAILDEADLOCK                 EVENTMONITORS
THERESAX   U         SAMPSEQUENCE         THERESAX     SEQUENCE
THERESAX   U         SQLE0F00             NULLID       PACKAGE
```

```
...
THERESAX   U         HI_OBJ_UNIQ          SYSTOOLS     TABLE CONSTRAINT

  257 record(s) selected.
```

Information returned

Table 156. Information returned by the OBJECTOWNERS administrative view

Column name	Data type	Description
OWNER	VARCHAR(128)	Authorization ID that owns this object.
OWNERTYPE	VARCHAR(1)	Authorization ID type: • U: user
OBJECTNAME	VARCHAR(128)	Database object name.
OBJECTSCHEMA	VARCHAR(128)	Database object schema.
OBJECTTYPE	VARCHAR(24)	Database object type.

PRIVILEGES administrative view – Retrieve privilege information

The PRIVILEGES administrative view returns all explicit privileges for all authorization IDs defined in the system catalogs from the currently connected database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the PRIVILEGES administrative view
- CONTROL privilege on the PRIVILEGES administrative view
- DATAACCESS authority

Example

Retrieve the privilege granted along with the object name, schema and type, for all authorization IDs.

```
SELECT AUTHID, PRIVILEGE, OBJECTNAME, OBJECTSCHEMA, OBJECTTYPE
   FROM SYSIBMADM.PRIVILEGES
```

The following is an example of output for this query.

```
AUTHID       PRIVILEGE   OBJECTNAME                OBJECTSCHEMA     OBJECTTYPE
-------...- ----------- ------------------...- -----------...- ----------...-
JESSICAE     EXECUTE     SQLE0F00                  NULLID           PACKAGE
PUBLIC       EXECUTE     SYSSH201                  NULLID           PACKAGE
JESSICAE     EXECUTE     SYSSH202                  NULLID           PACKAGE
PUBLIC       EXECUTE     SYSSH202                  NULLID           PACKAGE
DOCTOR       EXECUTE     PKG0123                   NULLID           PACKAGE
...
PUBLIC       EXECUTE     SQL051109185227800        SYSPROC          FUNCTION
JESSICAE     EXECUTE     SQL051109185227801        SYSPROC          FUNCTION
PUBLIC       EXECUTE     SQL051109185227801        SYSPROC          FUNCTION
JESSICAE     EXECUTE     SQL051109185227838        SYSPROC          FUNCTION
PUBLIC       EXECUTE     SQL051109185227838        SYSPROC          FUNCTION
...
PUBLIC       EXECUTE     LIST_SRVR_TYPES           SYSPROC          PROCEDURE
```

```
PUBLIC       EXECUTE     LIST_SRVR_VERSIONS      SYSPROC          PROCEDURE
PUBLIC       EXECUTE     LIST_WRAP_OPTIONS       SYSPROC          PROCEDURE
PUBLIC       EXECUTE     LIST_SRVR_OPTIONS       SYSPROC          PROCEDURE
...
SYSTEM                   POLICY_UNQ              SYSTOOLS         INDEX
PUBLIC       CREATEIN                            NULLID           SCHEMA
PUBLIC       UPDATE      COLUMNS                 SYSSTAT          VIEW
PUBLIC       UPDATE      COLGROUPS               SYSSTAT          VIEW
...
```

Information returned

Table 157. Information returned by the PRIVILEGES administrative view

Column name	Data type	Description
AUTHID	VARCHAR(128)	Authorization ID that has been explicitly granted this privilege.
AUTHIDTYPE	CHAR(1)	Authorization ID type: • U: user • R: role • G: group
PRIVILEGE	VARCHAR(11)	Privilege that has been explicitly granted to this authorization ID.
GRANTABLE	VARCHAR(1)	Indicates if the privilege is grantable: • Y: Grantable • N: Not grantable
OBJECTNAME	VARCHAR(128)	Database object name.
OBJECTSCHEMA	VARCHAR(128)	Database object schema.
OBJECTTYPE	VARCHAR(24)	Database object type.

Chapter 15. Snapshot routines and views

APPL_PERFORMANCE administrative view - Retrieve percentage of rows selected for an application

The APPL_PERFORMANCE administrative view displays information about the percentage of rows selected by an application. The information returned is for all database partitions for the currently connected database. This view can be used to look for applications that might be performing large table scans or to look for potentially troublesome queries.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the APPL_PERFORMANCE administrative view
- CONTROL privilege on the APPL_PERFORMANCE administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the report on application performance.

```
SELECT SNAPSHOT_TIMESTAMP, SUBSTR(AUTHID,1,10) AS AUTHID,
   SUBSTR(APPL_NAME,1,10) AS APPL_NAME,AGENT_ID,
   PERCENT_ROWS_SELECTED, DBPARTITIONNUM
   FROM SYSIBMADM.APPL_PERFORMANCE
```

The following is an example of output for this query.

```
SNAPSHOT_TIMESTAMP         AUTHID     APPL_NAME ...
-------------------------- ---------- ---------- ...
2006-01-07-17.01.15.966668 JESSICAE   db2bp.exe  ...
2006-01-07-17.01.15.980278 JESSICAE   db2taskd   ...
2006-01-07-17.01.15.980278 JESSICAE   db2bp.exe  ...
                                                 ...
   3 record(s) selected.                         ...
```

Output for this query (continued).

```
... AGENT_ID      PERCENT_ROWS_SELECTED DBPARTITIONNUM
... --------...-- --------------------- --------------
...           67                     -              1
...           68                     -              0
...           67                 57.14              0
...
```

Information returned

Table 158. Information returned by the APPL_PERFORMANCE administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AUTHID	VARCHAR(128)	auth_id - Authorization ID
APPL_NAME	VARCHAR(256)	appl_name - Application name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
PERCENT_ROWS_SELECTED	DECIMAL(5,2)	The percent of rows read from disk that were actually returned to the application. **Note:** The percentage shown will not be greater than 100.00 percent.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

APPLICATIONS administrative view - Retrieve connected database application information

The APPLICATIONS administrative view returns information on connected database applications. The view is an SQL interface for the LIST APPLICATIONS SHOW DETAIL CLP command, but only for the currently connected database. Its information is based on the SNAPAPPL_INFO administrative view.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the APPLICATIONS administrative view
- CONTROL privilege on the APPLICATIONS administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Example 1: List information for all the active applications in the single-partitioned database SAMPLE.

```
SELECT AGENT_ID, SUBSTR(APPL_NAME,1,10) AS APPL_NAME, AUTHID,
   APPL_STATUS FROM SYSIBMADM.APPLICATIONS WHERE DB_NAME = 'SAMPLE'
```

The following is an example of output for this query.

```
AGENT_ID             APPL_NAME  AUTHID    APPL_STATUS
-------------------- ---------- ---------- --------------------
                  23 db2bp.exe  JESSICAE   UOWEXEC

  1 record(s) selected.
```

Example 2: List the number of agents per application on database partition 0 for the multi-partition database SAMPLE.

```
SELECT SUBSTR(APPL_NAME, 1, 10) AS APPL_NAME, COUNT(*) AS NUM
   FROM SYSIBMADM.APPLICATIONS WHERE DBPARTITIONNUM = 0
   AND DB_NAME = 'SAMPLE' GROUP BY APPL_NAME
```

The following is an example of output for this query.

```
APPL_NAME  NUM
---------- -----------
db2bp.exe            3
javaw.exe            1

  2 record(s) selected.
```

Usage notes

The view does not support the GLOBAL syntax available from the CLP. However, aggregation can be done using SQL aggregation functions as data from all database partitions is returned from the view.

Information returned

Table 159. Information returned by the APPLICATIONS administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
CLIENT_DB_ALIAS	VARCHAR(128)	client_db_alias - Database alias used by application
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
APPL_NAME	VARCHAR(256)	appl_name - Application name
AUTHID	VARCHAR(128)	auth_id - Authorization ID
APPL_ID	VARCHAR(128)	appl_id - Application ID

Table 159. Information returned by the APPLICATIONS administrative view (continued)

Column name	Data type	Description or corresponding monitor element
APPL_STATUS	VARCHAR(22)	appl_status - Application status. This interface returns a text identifier based on defines in sqlmon.h, and is one of: • BACKUP • COMMIT_ACT • COMP • CONNECTED • CONNECTPEND • CREATE_DB • DECOUPLED • DISCONNECTPEND • INTR • IOERROR_WAIT • LOAD • LOCKWAIT • QUIESCE_TABLESPACE • RECOMP • REMOTE_RQST • RESTART • RESTORE • ROLLBACK_ACT • ROLLBACK_TO_SAVEPOINT • TEND • THABRT • THCOMT • TPREP • UNLOAD • UOWEXEC • UOWWAIT • WAITFOR_REMOTE
STATUS_CHANGE_TIME	TIMESTAMP	status_change_time - Application status change time
SEQUENCE_NO	VARCHAR(4)	sequence_no - Sequence number
CLIENT_PRDID	VARCHAR(128)	client_prdid - Client product/version ID
CLIENT_PID	BIGINT	client_pid - Client process ID

Table 159. Information returned by the APPLICATIONS administrative view (continued)

Column name	Data type	Description or corresponding monitor element
CLIENT_PLATFORM	VARCHAR(12)	client_platform - Client operating platform. This interface returns a text identifier based on defines in sqlmon.h, and is one of: • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS • WINDOWS95

Table 159. Information returned by the APPLICATIONS administrative view (continued)

Column name	Data type	Description or corresponding monitor element
CLIENT_PROTOCOL	VARCHAR(10)	client_protocol - Client communication protocol. This interface returns a text identifier based on the defines in sqlmon.h, • CPIC • LOCAL • NETBIOS • NPIPE • TCPIP (for DB2 UDB) • TCPIP4 • TCPIP6
CLIENT_NNAME	VARCHAR(128)	The client_nname monitor element is deprecated. The value returned is not a valid value.
COORD_NODE_NUM	SMALLINT	coord_node - Coordinating node
COORD_AGENT_PID	BIGINT	coord_agent_pid - Coordinator agent
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
TPMON_CLIENT_USERID	VARCHAR(256)	tpmon_client_userid - TP monitor client user ID
TPMON_CLIENT_WKSTN	VARCHAR(256)	tpmon_client_wkstn - TP monitor client workstation name
TPMON_CLIENT_APP	VARCHAR(256)	tpmon_client_app - TP monitor client application name
TPMON_ACC_STR	VARCHAR(200)	tpmon_acc_str - TP monitor client accounting string
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

BP_HITRATIO administrative view - Retrieve bufferpool hit ratio information

The BP_HITRATIO administrative view returns bufferpool hit ratios, including total hit ratio, data hit ratio, XDA hit ratio and index hit ratio, for all bufferpools and all database partitions in the currently connected database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the BP_HITRATIO administrative view
- CONTROL privilege on the BP_HITRATIO administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a report for all bufferpools in the connected database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(BP_NAME,1,14) AS BP_NAME,
   TOTAL_HIT_RATIO_PERCENT, DATA_HIT_RATIO_PERCENT,
   INDEX_HIT_RATIO_PERCENT, XDA_HIT_RATIO_PERCENT, DBPARTITIONNUM
   FROM SYSIBMADM.BP_HITRATIO ORDER BY DBPARTITIONNUM
```

The following is an example of output for this query.

```
DB_NAME  BP_NAME        TOTAL_HIT_RATIO_PERCENT DATA_HIT_RATIO_PERCENT ...
-------- -------------- ----------------------- ---------------------- ...
TEST     IBMDEFAULTBP                    63.09                  68.94 ...
TEST     IBMSYSTEMBP4K                       -                      - ...
TEST     IBMSYSTEMBP8K                       -                      - ...
TEST     IBMSYSTEMBP16K                      -                      - ...
TEST     IBMSYSTEMBP32K                      -                      - ...
```

Output for this query (continued).

```
... INDEX_HIT_RATIO_PERCENT XDA_HIT_RATIO_PERCENT DBPARTITIONNUM
... ----------------------- ---------------------- --------------
...                   43.20                      -              0
...                       -                      -              0
...                       -                      -              0
...                       -                      -              0
...                       -                      -              0
```

Usage notes

The ratio of physical reads to total reads gives the hit ratio for the bufferpool. The lower the hit ratio, the more the data is being read from disk rather than the cached buffer pool which can be a more costly operation.

Information returned

Table 160. Information returned by the BP_HITRATIO administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	Timestamp when the report was requested.
DB_NAME	VARCHAR(128)	db_name - Database name
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
TOTAL_LOGICAL_READS	BIGINT	Total logical reads (index, XDA and data) in the bufferpool.
TOTAL_PHYSICAL_READS	BIGINT	Total physical reads (index, XDA and data) in the bufferpool.
TOTAL_HIT_RATIO_PERCENT	DECIMAL(5,2)	Total hit ratio (index, XDA and data reads).

Table 160. Information returned by the BP_HITRATIO administrative view (continued)

Column name	Data type	Description or corresponding monitor element
DATA_LOGICAL_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
DATA_PHYSICAL_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
DATA_HIT_RATIO_PERCENT	DECIMAL(5,2)	Data hit ratio.
INDEX_LOGICAL_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
INDEX_PHYSICAL_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
INDEX_HIT_RATIO_PERCENT	DECIMAL(5,2)	Index hit ratio.
XDA_LOGICAL_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
XDA_PHYSICAL_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
XDA_HIT_RATIO_PERCENT	DECIMAL(5,2)	Auxiliary storage objects hit ratio.
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.

BP_READ_IO administrative view - Retrieve bufferpool read performance information

The BP_READ_IO administrative view returns bufferpool read performance information. This view can be used to look at each bufferpool to see how effective the prefetchers are.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the BP_READ_IO administrative view
- CONTROL privilege on the BP_READ_IO administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve total physical reads and average read time for all bufferpools on all partitions of the currently connected database.

```
SELECT SUBSTR(BP_NAME, 1, 15) AS BP_NAME, TOTAL_PHYSICAL_READS,
   AVERAGE_READ_TIME_MS, DBPARTITIONNUM
   FROM SYSIBMADM.BP_READ_IO ORDER BY DBPARTITIONNUM
```

The following is an example of output for this query.

```
BP_NAME         TOTAL_PHYSICAL_READS AVERAGE_READ_TIME_MS DBPARTITIONNUM
--------------- -------------------- -------------------- --------------
IBMDEFAULTBP                    811                    4              0
IBMSYSTEMBP4K                     0                    -              0
IBMSYSTEMBP8K                     0                    -              0
IBMSYSTEMBP16K                    0                    -              0
IBMDEFAULTBP                     34                    0              1
IBMSYSTEMBP4K                     0                    -              1
IBMSYSTEMBP8K                     0                    -              1
IBMDEFAULTBP                     34                    0              2
IBMSYSTEMBP4K                     0                    -              2
IBMSYSTEMBP8K                     0                    -              2

   10 record(s) selected.
```

Information returned

Table 161. Information returned by the BP_READ_IO administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	Date and time the report was generated.
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
TOTAL_PHYSICAL_READS	BIGINT	Total physical reads.
AVERAGE_READ_TIME_MS	BIGINT	Average read time in milliseconds.
TOTAL_ASYNC_READS	BIGINT	Total asynchronous reads.
AVERAGE_ASYNC_READ_TIME_MS	BIGINT	Average asynchronous read time in milliseconds.
TOTAL_SYNC_READS	BIGINT	Total synchronous reads.
AVERAGE_SYNC_READ_TIME_MS	BIGINT	Average synchronous read time in milliseconds.
PERCENT_SYNC_READS	DECIMAL(5,2)	Percentage of pages read synchronously without prefetching. If many of the applications are reading data synchronously without prefetching then the system might not be tuned optimally.

Table 161. Information returned by the BP_READ_IO administrative view (continued)

Column name	Data type	Description or corresponding monitor element
ASYNC_NOT_READ_PERCENT	DECIMAL(5,2)	Percentage of pages read asynchronously from disk, but never accessed by a query. If too many pages are read asynchronously from disk into the bufferpool, but no query ever accesses those pages, then the prefetching might degrade performance.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

BP_WRITE_IO administrative view - Retrieve bufferpool write performance information

The BP_WRITE_IO administrative view returns bufferpool write performance information per bufferpool.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the BP_WRITE_IO administrative view
- CONTROL privilege on the BP_WRITE_IO administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve total writes and average write time for all bufferpools on all database partitions of the currently connected database.

```
SELECT SUBSTR(BP_NAME, 1, 15) AS BP_NAME, TOTAL_WRITES,
   AVERAGE_WRITE_TIME_MS, DBPARTITIONNUM
   FROM SYSIBMADM.BP_WRITE_IO ORDER BY DBPARTITIONNUM
```

The following is an example of output for this query.

```
BP_NAME         TOTAL_WRITES     AVERAGE_WRITE_TIME_MS DBPARTITIONNUM
--------------- ------------...- --------------------- --------------
IBMDEFAULTBP                 11                     5              0
IBMSYSTEMBP4K                 0                     -              0
IBMSYSTEMBP8K                 0                     -              0
IBMSYSTEMBP16K                0                     -              0
IBMSYSTEMBP32K                0                     -              0
IBMDEFAULTBP                  0                     -              1
IBMSYSTEMBP4K                 0                     -              1
IBMSYSTEMBP8K                 0                     -              1
IBMDEFAULTBP                  0                     -              2
IBMSYSTEMBP4K                 0                     -              2
IBMSYSTEMBP8K                 0                     -              2

  11 record(s) selected.
```

Information returned

Table 162. Information returned by the BP_WRITE_IO administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time the report was generated.
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
TOTAL_WRITES	BIGINT	Total writes.
AVERAGE_WRITE_TIME_MS	BIGINT	Average write time in milliseconds.
TOTAL_ASYNC_WRITES	BIGINT	Total asynchronous writes.
PERCENT_WRITES_ASYNC	BIGINT	Percent of writes that are asynchronous.
AVERAGE_ASYNC_WRITE_TIME_MS	BIGINT	Average asynchronous write time in milliseconds.
TOTAL_SYNC_WRITES	BIGINT	Total synchronous writes.
AVERAGE_SYNC_WRITE_TIME_MS	BIGINT	Average synchronous write time in milliseconds.
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.

CONTAINER_UTILIZATION administrative view - Retrieve table space container and utilization information

The CONTAINER_UTILIZATION administrative view returns information about table space containers and utilization rates. It retrieve a similar report to the LIST TABLESPACES command on a single partitioned database. Its information is based on the SNAPCONTAINER administrative view.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the CONTAINER_UTILIZATION administrative view
- CONTROL privilege on the CONTAINER_UTILIZATION administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of all table spaces containers in the connected single partition database, including information on total and usable pages as well as their accessibility status.

```
SELECT SUBSTR(TBSP_NAME,1,20) AS TBSP_NAME, INT(TBSP_ID) AS TBSP_ID,
   SUBSTR(CONTAINER_NAME,1,45) AS CONTAINER_NAME, INT(CONTAINER_ID)
   AS CONTAINER_ID, CONTAINER_TYPE, INT(TOTAL_PAGES) AS TOTAL_PAGES,
   INT(USABLE_PAGES) AS USABLE_PAGES, ACCESSIBLE
   FROM SYSIBMADM.CONTAINER_UTILIZATION
```

The following is an example of output for this query.

```
TBSP_NAME            TBSP_ID     CONTAINER_NAME                              ...
---------------...- ----------- ------------------------------------...-- ...
SYSCATSPACE                    0 D:\DB2\NODE0000\SQL00001\SQLT0000.0        ...
TEMPSPACE1                     1 D:\DB2\NODE0000\SQL00001\SQLT0001.0        ...
USERSPACE1                     2 D:\DB2\NODE0000\SQL00001\SQLT0002.0        ...
SYSTOOLSPACE                   3 D:\DB2\NODE0000\SQL00001\SYSTOOLSPACE      ...
SYSTOOLSTMPSPACE               4 D:\DB2\NODE0000\SQL00001\SYSTOOLSTMPSPACE  ...

  5 record(s) selected.
```

Output for this query (continued).

```
... CONTAINER_ID CONTAINER_TYPE TOTAL_PAGES USABLE_PAGES ACCESSIBLE
... ------------ -------------- ----------- ------------ ----------
...            0 PATH                    0            0          1
...            0 PATH                    0            0          1
...            0 PATH                    0            0          1
...            0 PATH                    0            0          1
...            0 PATH                    0            0          1
```

Information returned

The BUFFERPOOL snapshot monitor switch must be enabled at the database manager configuration for the file system information to be returned.

Table 163. Information returned by the CONTAINER_UTILIZATION administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
CONTAINER_NAME	VARCHAR(256)	container_name - Container name
CONTAINER_ID	BIGINT	container_id - Container identification
CONTAINER_TYPE	VARCHAR(16)	container_type - Container type This is a text identifier based on the defines in sqlutil.h and is one of: • DISK_EXTENT_TAG • DISK_PAGE_TAG • FILE_EXTENT_TAG • FILE_PAGE_TAG • PATH
TOTAL_PAGES	BIGINT	container_total_pages - Total pages in container
USABLE_PAGES	BIGINT	container_usable_pages - Usable pages in container
ACCESSIBLE	SMALLINT	container_accessible - Accessibility of container
STRIPE_SET	BIGINT	container_stripe_set - Stripe set
FS_ID	VARCHAR(22)	fs_id - Unique file system identification number
FS_TOTAL_SIZE_KB	BIGINT	fs_total_size - Total size of a file system. This interface returns the value in KB.
FS_USED_SIZE_KB	BIGINT	fs_used_size - Amount of space used on a file system. This interface returns the value in KB.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

LOCKS_HELD administrative view - Retrieve information on locks held

Note: This administrative view has been deprecated and replaced by the "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415, "MON_GET_LOCKS - list all locks in the currently connected database" on page 442, and "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381.

The LOCKS_HELD administrative view returns information on current locks held.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the LOCKS_HELD administrative view
- CONTROL privilege on the LOCKS_HELD administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Example 1: List the total number of locks held by each table in the database SAMPLE.

```
SELECT TABSCHEMA, TABNAME, COUNT(*) AS NUMBER_OF_LOCKS_HELD
   FROM SYSIBMADM.LOCKS_HELD WHERE DB_NAME = 'SAMPLE'
   GROUP BY DBPARTITIONNUM, TABSCHEMA, TABNAME
```

The following is an example of output for this query.

```
TABSCHEMA      TABNAME       NUMBER_OF_LOCKS_HELD
----------...- ---------...- --------------------
JESSICAE       EMPLOYEE                         5
JESSICAE       EMP_RESUME                       1
JESSICAE       ORG                              3
```

Example 2: List all the locks that have not escalated in the currently connected database, SAMPLE.

```
SELECT AGENT_ID, TABSCHEMA, TABNAME, LOCK_OBJECT_TYPE, LOCK_MODE,
   LOCK_STATUS FROM SYSIBMADM.LOCKS_HELD WHERE LOCK_ESCALATION = 0
   AND DBPARTITIONNUM = 0
```

The following is an example of output for this query.

```
AGENT_ID     TABSCHEMA     TABNAME      LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS
--------...- ---------...- ---------...- ------------------ ---------- -----------
         680 JESSICAE      EMPLOYEE     INTERNALV_LOCK     S          GRNT
         680 JESSICAE      EMPLOYEE     INTERNALP_LOCK     S          GRNT
```

Example 3: List lock information for the locks that are currently held by the application with agent ID 310.

```
SELECT TABSCHEMA, TABNAME, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS,
   LOCK_ESCALATION FROM SYSIBMADM.LOCKS_HELD WHERE AGENT_ID = 310
```

The following is an example of output for this query.

```
TABSCHEMA     TABNAME      LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS
---------...- --------...- ------------------ ---------- -----------
JESSICAE      EMP_RESUME   TABLE_LOCK         S          GRNT
JESSICAE      EMPLOYEE     ROW_LOCK           S          GRNT
```

Information returned

Table 164. Information returned by the LOCKS_HELD administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	Date and time the report was generated.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
APPL_NAME	VARCHAR(256)	appl_name - Application name
AUTHID	VARCHAR(128)	auth_id - Authorization ID
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
TAB_FILE_ID	BIGINT	table_file_id - Table file identification
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
LOCK_NAME	VARCHAR(32)	lock_name - Lock name

Table 164. Information returned by the LOCKS_HELD administrative view (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_STATUS	VARCHAR(10)	lock_status - Lock status. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • CONV • GRNT
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

LOCKWAITS administrative view - Retrieve current lockwaits information

Note: This administrative view has been deprecated and replaced by the "MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503.

The LOCKWAITS administrative view returns information about DB2 agents working on behalf of applications that are waiting to obtain locks.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the LOCKWAITS administrative view
- CONTROL privilege on the LOCKWAITS administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON

- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Example 1: List information for all the lock waits for application with agent ID 89.

```
SELECT SUBSTR(TABSCHEMA,1,8) AS TABSCHEMA, SUBSTR(TABNAME,1,15) AS TABNAME,
   LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_MODE_REQUESTED, AGENT_ID_HOLDING_LK
   FROM SYSIBMADM.LOCKWAITS WHERE AGENT_ID = 89
```

The following is an example of output for this query.

```
TABSCHEMA TABNAME     LOCK_OBJECT_TYPE LOCK_MODE  ...
--------- -------...- --------------- ---------- ...
JESSICAE  T1          ROW_LOCK         X          ...

  1 record(s) selected.
```

Output for this query (continued).

```
... LOCK_MODE_REQUESTED AGENT_ID_HOLDING_LK
... ------------------- --------------------
... NS                                    7
```

Example 2: List the total number of outstanding lock requests per table in the database SAMPLE. By sorting the output by number of requests, tables with the highest contention can be identified.

```
SELECT SUBSTR(TABSCHEMA,1,8) AS TABSCHEMA, SUBSTR(TABNAME, 1, 15)
   AS TABNAME, COUNT(*) AS NUM_OF_LOCK_REQUESTS_WAITING,
   DBPARTITIONNUM
   FROM SYSIBMADM.LOCKWAITS WHERE DB_NAME = 'SAMPLE'
   GROUP BY TABSCHEMA, TABNAME, DBPARTITIONNUM
   ORDER BY NUM_OF_LOCK_REQUESTS_WAITING DESC
```

The following is an example of output for this query.

```
TABSCHEMA TABNAME     NUM_OF_LOCK_REQUESTS_WAITING DBPARTITIONNUM
--------- -------...- ---------------------------- --------------
JESSICAE  T3                                    2              0
JESSICAE  T1                                    1              0
JESSICAE  T2                                    1              0

  3 record(s) selected.
```

Information returned

Table 165. Information returned by the LOCKWAITS administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	Date and time the report was generated.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
APPL_NAME	VARCHAR(256)	appl_name - Application name
AUTHID	VARCHAR(128)	auth_id - Authorization ID
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name

Table 165. Information returned by the LOCKWAITS administrative view (continued)

Column name	Data type	Description or corresponding monitor element
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
SUBSECTION_NUMBER	BIGINT	ss_number - Subsection number
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
LOCK_WAIT_START_TIME	TIMESTAMP	lock_wait_start_time - Lock wait start timestamp
LOCK_NAME	VARCHAR(32)	lock_name - Lock name

Table 165. Information returned by the LOCKWAITS administrative view (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_MODE_REQUESTED	VARCHAR(10)	lock_mode_requested - Lock mode requested. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
AGENT_ID_HOLDING_LK	BIGINT	agent_id_holding_lock - Agent ID holding lock
APPL_ID_HOLDING_LK	VARCHAR(128)	appl_id_holding_lk - Application ID holding lock
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

LOG_UTILIZATION administrative view - Retrieve log utilization information

The LOG_UTILIZATION administrative view returns information about log utilization for the currently connected database. A single row is returned for each database partition.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the LOG_UTILIZATION administrative view
- CONTROL privilege on the LOG_UTILIZATION administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

List the log utilization for the currently connected database, SAMPLE.

```
SELECT * FROM SYSIBMADM.LOG_UTILIZATION
```

The following is an example of output for this query.

```
DB_NAME  ... LOG_UTILIZATION_PERCENT TOTAL_LOG_USED_KB    ...
-------- ... ---------------------- -------------------- ...
SAMPLE   ...                   9.75                 1989 ...
                                                         ...
  1 record(s) selected.                                  ...
```

Output for this query (continued).

```
... TOTAL_LOG_AVAILABLE_KB TOTAL_LOG_USED_TOP_KB DBPARTITIONNUM
... ---------------------- --------------------- --------------
...                  18411                  1990              0
...
...
```

Usage note

For databases that are configured for infinite logging, the LOG_UTILIZATION_PERCENT and TOTAL_LOG_AVAILABLE_KB will be NULL.

Information returned

Table 166. Information returned by the LOG_UTILIZATION administrative view

Column name	Data type	Description or corresponding monitor element
DB_NAME	VARCHAR(128)	db_name - Database name
LOG_UTILIZATION_PERCENT	DECIMAL(5,2)	Percent utilization of total log space.
TOTAL_LOG_USED_KB	BIGINT	total_log_used - Total log space used. This interface returns the value in KB.
TOTAL_LOG_AVAILABLE_KB	BIGINT	total_log_available - Total log available. This interface returns the value in KB.

Table 166. Information returned by the LOG_UTILIZATION administrative view (continued)

Column name	Data type	Description or corresponding monitor element
TOTAL_LOG_USED_TOP_KB	BIGINT	tot_log_used_top - Maximum total log space used. This interface returns the value in KB.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

LONG_RUNNING_SQL administrative view

The LONG_RUNNING_SQL administrative view returns the longest running SQL statements in the currently connected database.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the LONG_RUNNING_SQL administrative view
- CONTROL privilege on the LONG_RUNNING_SQL administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a report on long running SQL statements in the currently connected database.

```
SELECT SUBSTR(STMT_TEXT, 1, 50) AS STMT_TEXT, AGENT_ID,
   ELAPSED_TIME_MIN, APPL_STATUS, DBPARTITIONNUM
   FROM SYSIBMADM.LONG_RUNNING_SQL ORDER BY DBPARTITIONNUM
```

The following is an example of output for this query.

```
STMT_TEXT                         AGENT_ID     ...
----------------------------...- --------...- ...
select * from dbuser.employee              228 ...
select * from dbuser.employee              228 ...
select * from dbuser.employee              228 ...
                                               ...
3 record(s) selected.                          ...
```

Output for this query (continued).

```
... ELAPSED_TIME_MIN APPL_STATUS     DBPARTITIONNUM
... ---------------- -----------...- --------------
...                2 UOWWAIT                      0
...                0 CONNECTED                    1
...                0 CONNECTED                    2
```

Usage note

This view can be used to identify long-running SQL statements in the database. You can look at the currently running queries to see which statements are the longest running and the current status of the query. Further investigation can be done of the application containing the SQL statement, using agent ID as the unique identifier. If executing a long time and waiting on a lock, you might want to dig deeper using the LOCKWAITS or LOCKS_HELD administrative views. If "waiting on User", this means that the DB2 server is not doing anything but rather is waiting for the application to do something (like issue the next fetch or submit the next SQL statement).

Information returned

Table 167. Information returned by the LONG_RUNNING_SQL administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	Time the report was generated.
ELAPSED_TIME_MIN	INTEGER	Elapsed time of the statement in minutes.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
APPL_NAME	VARCHAR(256)	appl_name - Application name

Table 167. Information returned by the LONG_RUNNING_SQL administrative view (continued)

Column name	Data type	Description or corresponding monitor element
APPL_STATUS	VARCHAR(22)	appl_status - Application status. This interface returns a text identifier based on the defines in sqlmon.h, and is one of: • BACKUP • COMMIT_ACT • COMP • CONNECTED • CONNECTPEND • CREATE_DB • DECOUPLED • DISCONNECTPEND • INTR • IOERROR_WAIT • LOAD • LOCKWAIT • QUIESCE_TABLESPACE • RECOMP • REMOTE_RQST • RESTART • RESTORE • ROLLBACK_ACT • ROLLBACK_TO_SAVEPOINT • TEND • THABRT • THCOMT • TPREP • UNLOAD • UOWEXEC • UOWWAIT • WAITFOR_REMOTE
AUTHID	VARCHAR(128)	auth_id - Authorization ID
INBOUND_COMM_ADDRESS	VARCHAR(32)	inbound_comm_address - Inbound communication address
STMT_TEXT	CLOB(16 M)	stmt_text - SQL statement text
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

QUERY_PREP_COST administrative view - Retrieve statement prepare time information

The QUERY_PREP_COST administrative view returns a list of statements with information about the time required to prepare the statement.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the QUERY_PREP_COST administrative view
- CONTROL privilege on the SNAPAGENT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAPDYN_SQL table function
- DATAACCESS authority

Example

Retrieve a report on the queries with the highest percentage of time spent on preparing.

```
SELECT NUM_EXECUTIONS, AVERAGE_EXECUTION_TIME_S, PREP_TIME_PERCENT,
   SUBSTR(STMT_TEXT, 1, 30) AS STMT_TEXT, DBPARTITIONNUM
   FROM SYSIBMADM.QUERY_PREP_COST ORDER BY PREP_TIME_PERCENT
```

The following is an example of output for this query.

```
NUM_EXECUTIONS     AVERAGE_EXECUTION_TIME_S ...
--------------...- ------------------------ ...
                1                       25 ...

  1 record(s) selected.
```

Output for this query (continued).

```
... PREP_TIME_PERCENT STMT_TEXT                     DBPARTITIONNUM
... ---------------- ------------------------------ --------------
...              0.0 select * from dbuser.employee               0
```

Usage notes

When selecting from the view, an order by clause can be used to identify queries with the highest prep cost. You can examine this view to see how frequently a query is run as well as the average execution time for each of these queries. If the time it takes to compile and optimize a query is almost as long as it takes for the query to execute, you might want to look at the optimization class that you are using. Lowering the optimization class might make the query complete optimization more rapidly and therefore return a result sooner. However, if a query takes a significant amount of time to prepare yet is executed thousands of times (without being prepared again) then the optimization class might not be an issue.

Information returned

Table 168. Information returned by the QUERY_PREP_COST administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time the report was generated.
NUM_EXECUTIONS	BIGINT	num_executions - Statement executions
AVERAGE_EXECUTION_TIME_S	BIGINT	Average execution time (in seconds).
PREP_TIME_MS	BIGINT	prep_time_worst - Statement worst preparation time (in milliseconds).
PREP_TIME_PERCENT	DECIMAL(5,2)	Percent of execution time spent on preparation.
STMT_TEXT	CLOB(2 M)	stmt_text - SQL statement text
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPAGENT administrative view and SNAP_GET_AGENT table function – Retrieve agent logical data group application snapshot information

The SNAPAGENT administrative view and the SNAP_GET_AGENT table function return information about agents from an application snapshot, in particular, the agent logical data group.

SNAPAGENT administrative view

This administrative view allows you to retrieve agent logical data group application snapshot information for the currently connected database.

Used with the SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPAPPL_INFO, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAGENT administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 169 on page 576 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPAGENT administrative view
- CONTROL privilege on the SNAPAGENT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_AGENT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve all application snapshot information for the currently connected database from the agent logical data group.

```
SELECT * FROM SYSIBMADM.SNAPAGENT
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         DB_NAME     AGENT_ID        ...
-------------------------- -------...- ---------...--- ...
2005-07-19-11.03.26.740423 SAMPLE                  101 ...
2005-07-19-11.03.26.740423 SAMPLE                   49 ...
                                                       ...
  2 record(s) selected.                                ...
```

Output from this query (continued).

```
...  AGENT_PID            LOCK_TIMEOUT_VAL     DBPARTITIONNUM
...  -------------------- -------------------- --------------
...                11980                   -1              0
...                15940                   -1              0
...
...
```

SNAP_GET_AGENT table function

The SNAP_GET_AGENT table function returns the same information as the SNAPAGENT administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_AGENT table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 169 on page 576 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_AGENT table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_AGENT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve all application snapshot information for all applications in all active databases.

```
SELECT * FROM TABLE(SNAP_GET_AGENT(CAST(NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         DB_NAME     AGENT_ID      ...
-------------------------- -------...- --------...-- ...
2006-01-03-17.21.38.530785 SAMPLE                48 ...
2006-01-03-17.21.38.530785 SAMPLE                47 ...
2006-01-03-17.21.38.530785 SAMPLE                46 ...
2006-01-03-17.21.38.530785 TESTDB                30 ...
2006-01-03-17.21.38.530785 TESTDB                29 ...
2006-01-03-17.21.38.530785 TESTDB                28 ...

6 record(s) selected.
```

Output from this query (continued).

```
... AGENT_PID        LOCK_TIMEOUT_VAL     DBPARTITIONNUM
... ---------...---- -------------------- --------------
...             7696                   -1              0
...             8536                   -1              0
...             6672                   -1              0
...             2332                   -1              0
...             8360                   -1              0
...             6736                   -1              0
...
```

Information returned

Table 169. Information returned by the SNAPAGENT administrative view and the SNAP_GET_AGENT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
AGENT_PID	BIGINT	agent_pid - Engine dispatchable unit (EDU)
LOCK_TIMEOUT_VAL	BIGINT	lock_timeout_val - Lock timeout (seconds)
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.

SNAPAGENT_MEMORY_POOL administrative view and SNAP_GET_AGENT_MEMORY_POOL table function – Retrieve memory_pool logical data group snapshot information

The SNAPAGENT_MEMORY_POOL administrative view and the SNAP_GET_AGENT_MEMORY_POOL table function return information about memory usage at the agent level.

SNAPAGENT_MEMORY_POOL administrative view

This administrative view allows you to retrieve the memory_pool logical data group snapshot information about memory usage at the agent level for the currently connected database.

Used with the SNAPAGENT, SNAPAPPL, SNAPAPPL_INFO, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAGENT_MEMORY_POOL administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 170 on page 579 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:
- SELECT privilege on the SNAPAGENT_MEMORY_POOL administrative view
- CONTROL privilege on the SNAPAGENT_MEMORY_POOL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the SNAP_GET_AGENT_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size.

```
SELECT AGENT_ID, POOL_ID, POOL_CUR_SIZE FROM SYSIBMADM.SNAPAGENT_MEMORY_POOL
```

The following is an example of output from this query.

```
AGENT_ID       POOL_ID POOL_  CUR_SIZE
--------...--- -------------- ---------...------
            48 APPLICATION                65536
            48 OTHER                      65536
            48 APPL_CONTROL               65536
            47 APPLICATION                65536
            47 OTHER                     131072
            47 APPL_CONTROL               65536
            46 OTHER                     327680
            46 APPLICATION               262144
            46 APPL_CONTROL               65536

9 record(s) selected.
```

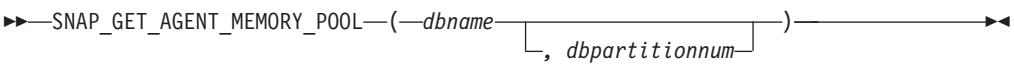
SNAP_GET_AGENT_MEMORY_POOL table function

The SNAP_GET_AGENT_MEMORY_POOL table function returns the same information as the SNAPAGENT_MEMORY_POOL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_AGENT_MEMORY_POOL table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command.

Refer to Table 170 on page 579 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_AGENT_MEMORY_POOL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_AGENT_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size for all databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID, POOL_ID, POOL_CUR_SIZE
   FROM TABLE(SNAP_GET_AGENT_MEMORY_POOL(CAST (NULL AS VARCHAR(128)), -1))
   AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID       POOL_ID        POOL_CUR_SIZE
-------- --------...--- -------------- --------------------
SAMPLE               48 APPLICATION                   65536
SAMPLE               48 OTHER                         65536
SAMPLE               48 APPL_CONTROL                  65536
SAMPLE               47 APPLICATION                   65536
SAMPLE               47 OTHER                        131072
SAMPLE               47 APPL_CONTROL                  65536
SAMPLE               46 OTHER                        327680
SAMPLE               46 APPLICATION                  262144
SAMPLE               46 APPL_CONTROL                  65536
TESTDB               30 APPLICATION                   65536
TESTDB               30 OTHER                         65536
TESTDB               30 APPL_CONTROL                  65536
TESTDB               29 APPLICATION                   65536
TESTDB               29 OTHER                        131072
TESTDB               29 APPL_CONTROL                  65536
TESTDB               28 OTHER                        327680
TESTDB               28 APPLICATION                   65536
TESTDB               28 APPL_CONTROL                  65536

18 record(s) selected.
```

Information returned

Table 170. Information returned by the SNAPAGENT_MEMORY_POOL administrative view and the SNAP_GET_AGENT_MEMORY_POOL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
AGENT_PID	BIGINT	agent_pid - Engine dispatchable unit (EDU)

Table 170. Information returned by the SNAPAGENT_MEMORY_POOL administrative view and the SNAP_GET_AGENT_MEMORY_POOL table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ID	VARCHAR(14)	pool_id - Memory pool identifier. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • APP_GROUP • APPL_CONTROL • APPLICATION • BP • CAT_CACHE • DATABASE • DFM • FCMBP • IMPORT_POOL • LOCK_MGR • MONITOR • OTHER • PACKAGE_CACHE • QUERY • SHARED_SORT • SORT • STATEMENT • STATISTICS • UTILITY
POOL_CUR_SIZE	BIGINT	pool_cur_size - Current size of memory pool
POOL_WATERMARK	BIGINT	pool_watermark - Memory pool watermark
POOL_CONFIG_SIZE	BIGINT	pool_config_size - Configured size of memory pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPAPPL_INFO administrative view and SNAP_GET_APPL_INFO_V95 table function - Retrieve appl_info logical data group snapshot information

The SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function return information about applications from an application snapshot, in particular, the appl_info logical data group.

SNAPAPPL_INFO administrative view

This administrative view allows you to retrieve appl_info logical data group snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAPPL_INFO administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 171 on page 583 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPAPPL_INFO administrative view
- CONTROL privilege on the SNAPAPPL_INFO administrative view
- DATAACCESS authority

Additionally, one of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_APPL_INFO_V95 table function
- DATAACCESS authority

Also, one of the following authorities is required:

- SYSMON
- SYSMAINT
- SYSCTRL
- SYSADM

Example

Retrieve the status of the applications connected to the current database.

```
SELECT AGENT_ID, SUBSTR(APPL_NAME,1,10) AS APPL_NAME, APPL_STATUS
   FROM SYSIBMADM.SNAPAPPL_INFO
```

The following is an example of output from this query.

```
AGENT_ID             APPL_NAME  APPL_STATUS
-------------------- ---------- ----------------------
                 101 db2bp.exe  UOWEXEC
                  49 db2bp.exe  CONNECTED

  2 record(s) selected.
```

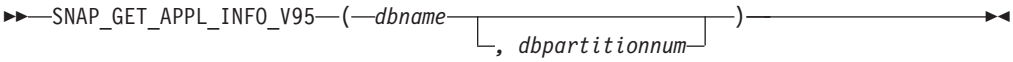
SNAP_GET_APPL_INFO_V95 table function

The SNAP_GET_APPL_INFO_V95 table function returns the same information as the SNAPAPPL_INFO administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_APPL_INFO_V95 table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 171 on page 583 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_APPL_INFO_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_APPL_INFO_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve the status of all applications on the connected database partition.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID,
   SUBSTR(APPL_NAME,1,10) AS APPL_NAME, APPL_STATUS
   FROM TABLE(SNAP_GET_APPL_INFO_V95(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID             APPL_NAME  APPL_STATUS
-------- -------------------- ---------- ----------------------
TOOLSDB                    14 db2bp.exe  CONNECTED
SAMPLE                     15 db2bp.exe  UOWEXEC
SAMPLE                      8 javaw.exe  CONNECTED
SAMPLE                      7 db2bp.exe  UOWWAIT

  4 record(s) selected.
```

The following shows what you obtain when you SELECT from the result of the table function.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AUTHORITY_LVL
   FROM TABLE(SNAP_GET_APPL_INFO_V95(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AUTHORITY_LVL
-------- -----------------------------------------------------....
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)

3 record(s) selected.
```

Information returned

Table 171. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)

Table 171. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
APPL_STATUS	VARCHAR(22)	appl_status - Application status. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • BACKUP • COMMIT_ACT • COMP • CONNECTED • CONNECTPEND • CREATE_DB • DECOUPLED • DISCONNECTPEND • INTR • IOERROR_WAIT • LOAD • LOCKWAIT • QUIESCE_TABLESPACE • RECOMP • REMOTE_RQST • RESTART • RESTORE • ROLLBACK_ACT • ROLLBACK_TO_SAVEPOINT • TEND • THABRT • THCOMT • TPREP • UNLOAD • UOWEXEC • UOWWAIT • WAITFOR_REMOTE
CODEPAGE_ID	BIGINT	codepage_id - ID of code page used by application
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
COORD_NODE_NUM	SMALLINT	coord_node - Coordinating node

Table 171. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
AUTHORITY_LVL	VARCHAR(512)	authority_bitmap - User authorization level. This interface returns a text identifier based on the database authorities defined in sql.h and their source, and has the following format: authority(source, ...) + authority(source, ...) + ... The source of an authority can be multiple: either from a USER, a GROUP, or a USER and a GROUP. Possible values for "authority": • ACCESSCTRL • BINDADD • CONNECT • CREATE_EXT_RT • CREATE_NOT_FENC • CREATETAB • DATAACCESS • DBADM • EXPLAIN • IMPLICIT_SCHEMA • LOAD • LIBADM • QUIESCE_CONN • SECADM • SQLADM • SYSADM • SYSCTRL • SYSMAINT • SYSMON • SYSQUIESCE • WLMADM Possible values for "source": • USER – authority granted to the user or to a role granted to the user. • GROUP – authority granted to a group to which the user belongs or to a role granted to the group to which the user belongs.
CLIENT_PID	BIGINT	client_pid - Client process ID
COORD_AGENT_PID	BIGINT	coord_agent_pid - Coordinator agent

Table 171. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
STATUS_CHANGE_TIME	TIMESTAMP	status_change_time - Application status change time
CLIENT_PLATFORM	VARCHAR(12)	client_platform - Client operating platform. This interface returns a text identifier based on the defines in `sqlmon.h`, • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS

Table 171. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
CLIENT_PROTOCOL	VARCHAR(10)	client_protocol - Client communication protocol. This interface returns a text identifier based on the defines in `sqlmon.h`, • CPIC • LOCAL • NETBIOS • NPIPE • TCPIP (for DB2 UDB) • TCPIP4 • TCPIP6
TERRITORY_CODE	SMALLINT	territory_code - Database territory code
APPL_NAME	VARCHAR(256)	appl_name - Application name
APPL_ID	VARCHAR(128)	appl_id - Application ID
SEQUENCE_NO	VARCHAR(4)	sequence_no - Sequence number
PRIMARY_AUTH_ID	VARCHAR(128)	auth_id - Authorization ID
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
CLIENT_NNAME	VARCHAR(128)	The client_nname monitor element is deprecated. The value returned is not a valid value.
CLIENT_PRDID	VARCHAR(128)	client_prdid - Client product/version ID
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
CLIENT_DB_ALIAS	VARCHAR(128)	client_db_alias - Database alias used by application
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
EXECUTION_ID	VARCHAR(128)	execution_id - User login ID
CORR_TOKEN	VARCHAR(128)	corr_token - DRDA correlation token
TPMON_CLIENT_USERID	VARCHAR(256)	tpmon_client_userid - TP monitor client user ID
TPMON_CLIENT_WKSTN	VARCHAR(256)	tpmon_client_wkstn - TP monitor client workstation name
TPMON_CLIENT_APP	VARCHAR(256)	tpmon_client_app - TP monitor client application name
TPMON_ACC_STR	VARCHAR(200)	tpmon_acc_str - TP monitor client accounting string
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.
WORKLOAD_ID	INTEGER	Current workload ID.

Table 171. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
IS_SYSTEM_APPL	SMALLINT	The value of IS_SYSTEM_APPL indicates whether or not the application is a DB2 internal system application: 0 means it is a user application 1 means it is a system application. An example of a DB2 system application is a DB2 event monitor. In general, the names of DB2 system applications begin with "db2". For example: db2stmm, db2taskd.

SNAPAPPL administrative view and SNAP_GET_APPL_V95 table function - Retrieve appl logical data group snapshot information

The "SNAPAPPL administrative view" and the "SNAP_GET_APPL_V95 table function" on page 589 return information about applications from an application snapshot, in particular, the appl logical data group.

SNAPAPPL administrative view

This administrative view allows you to retrieve appl logical data group snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL_INFO, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAPPL administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 172 on page 591 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPAPPL administrative view
- CONTROL privilege on the SNAPAPPL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_APPL_V95 table function

- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on rows read and written for each application in the connected database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID, ROWS_READ, ROWS_WRITTEN
   FROM SYSIBMADM.SNAPAPPL
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID             ROWS_READ            ROWS_WRITTEN
-------- -------------------- -------------------- --------------------
SAMPLE                     7                   25                    0

  1 record(s) selected.
```

SNAP_GET_APPL_V95 table function

The SNAP_GET_APPL_V95 table function returns the same information as the SNAPAPPL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_INFO_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_APPL_V95 table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 172 on page 591 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty

string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_APPL_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_APPL_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on rows read and written for each application for all active databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID, ROWS_READ, ROWS_WRITTEN
   FROM TABLE (SNAP_GET_APPL_V95(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID       ROWS_READ      ROWS_WRITTEN
-------- --------...--- ---------...-- ------------...-
WSDB                679              0                 0
WSDB                461              3                 0
WSDB                460              4                 0
TEST                680              4                 0
TEST                455              6                 0
TEST                454              0                 0
TEST                453             50                 0
```

Information returned

Table 172. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
UOW_LOG_SPACE_USED	BIGINT	uow_log_space_used - Unit of work log space used
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
INACT_STMTHIST_SZ	BIGINT	stmt_history_list_size - Statement history list size
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes

Table 172. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
TOTAL_SORTS	BIGINT	total_sorts - Total sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
COMMIT_SQL_STMTS	BIGINT	commit_sql_stmts - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	rollback_sql_stmts - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	dynamic_sql_stmts - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	static_sql_stmts - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	failed_sql_stmts - Failed statement operations
SELECT_SQL_STMTS	BIGINT	select_sql_stmts - Select SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements

Table 172. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
UID_SQL_STMTS	BIGINT	uid_sql_stmts - UPDATE/INSERT/DELETE SQL statements executed
INT_AUTO_REBINDS	BIGINT	int_auto_rebinds - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
INT_COMMITS	BIGINT	int_commits - Internal commits
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	int_deadlock_rollbacks - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_SELECTED	BIGINT	rows_selected - Rows selected
BINDS_PRECOMPILES	BIGINT	binds_precompiles - Binds/precompiles attempted
OPEN_REM_CURS	BIGINT	open_rem_curs - Open remote cursors
OPEN_REM_CURS_BLK	BIGINT	open_rem_curs_blk - Open remote cursors with blocking
REJ_CURS_BLK	BIGINT	rej_curs_blk - Rejected block cursor requests
ACC_CURS_BLK	BIGINT	acc_curs_blk - Accepted block cursor requests
SQL_REQS_SINCE_COMMIT	BIGINT	sql_reqs_since_commit - SQL requests since last commit
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
OPEN_LOC_CURS	BIGINT	open_loc_curs - Open local cursors
OPEN_LOC_CURS_BLK	BIGINT	open_loc_curs_blk - Open local cursors with blocking
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts

Table 172. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
CAT_CACHE_OVERFLOWS	BIGINT	cat_cache_overflows - Catalog cache overflows
NUM_AGENTS	BIGINT	num_agents - Number of agents working on a statement
AGENTS_STOLEN	BIGINT	agents_stolen - Stolen agents
ASSOCIATED_AGENTS_TOP	BIGINT	associated_agents_top - Maximum number of associated agents
APPL_PRIORITY	BIGINT	appl_priority - Application agent priority
APPL_PRIORITY_TYPE	VARCHAR(16)	appl_priority_type - Application priority type. This interface returns a text identifier, based on defines in `sqlmon.h`, and is one of: • DYNAMIC_PRIORITY • FIXED_PRIORITY
PREFETCH_WAIT_TIME	BIGINT	prefetch_wait_time - Time waited for prefetch
APPL_SECTION_LOOKUPS	BIGINT	appl_section_lookups - Section lookups
APPL_SECTION_INSERTS	BIGINT	appl_section_inserts - Section inserts
LOCKS_WAITING	BIGINT	locks_waiting - Current agents waiting on locks
TOTAL_HASH_JOINS	BIGINT	total_hash_joins - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	hash_join_overflows - Hash join overflows
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	hash_join_small_overflows - Hash join small overflows
APPL_IDLE_TIME	BIGINT	appl_idle_time - Application idle time
UOW_LOCK_WAIT_TIME	BIGINT	uow_lock_wait_time - Total time unit of work waited on locks
UOW_COMP_STATUS	VARCHAR(14)	uow_comp_status - Unit of work completion status. This interface returns a text identifier, based on defines in `sqlmon.h`, and is one of: • APPL_END • UOWABEND • UOWCOMMIT • UOWDEADLOCK • UOWLOCKTIMEOUT • UOWROLLBACK • UOWUNKNOWN

Table 172. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
AGENT_USR_CPU_TIME_S	BIGINT	agent_usr_cpu_time - User CPU time used by agent (in seconds)*
AGENT_USR_CPU_TIME_MS	BIGINT	agent_usr_cpu_time - User CPU time used by agent (fractional, in microseconds)*
AGENT_SYS_CPU_TIME_S	BIGINT	agent_sys_cpu_time - System CPU time used by agent (in seconds)*
AGENT_SYS_CPU_TIME_MS	BIGINT	agent_sys_cpu_time - System CPU time used by agent (fractional, in microseconds)*
APPL_CON_TIME	TIMESTAMP	appl_con_time - Connection request start timestamp
CONN_COMPLETE_TIME	TIMESTAMP	conn_complete_time - Connection request completion timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
UOW_START_TIME	TIMESTAMP	uow_start_time - Unit of work start timestamp
UOW_STOP_TIME	TIMESTAMP	uow_stop_time - Unit of work stop timestamp
PREV_UOW_STOP_TIME	TIMESTAMP	prev_uow_stop_time - Previous unit of work completion timestamp
UOW_ELAPSED_TIME_S	BIGINT	uow_elapsed_time - Most recent unit of work elapsed time (in seconds)*
UOW_ELAPSED_TIME_MS	BIGINT	uow_elapsed_time - Most recent unit of work elapsed time (fractional, in microseconds)*
ELAPSED_EXEC_TIME_S	BIGINT	elapsed_exec_time - Statement execution elapsed time (in seconds)*
ELAPSED_EXEC_TIME_MS	BIGINT	elapsed_exec_time - Statement execution elapsed time (fractional, in microseconds)*
INBOUND_COMM_ADDRESS	VARCHAR(32)	inbound_comm_address - Inbound communication address
LOCK_TIMEOUT_VAL	BIGINT	lock_timeout_val - Lock timeout (seconds)
PRIV_WORKSPACE_NUM_ OVERFLOWS	BIGINT	priv_workspace_num_overflows - Private workspace overflows
PRIV_WORKSPACE_SECTION_ INSERTS	BIGINT	priv_workspace_section_inserts - Private workspace section inserts
PRIV_WORKSPACE_SECTION_ LOOKUPS	BIGINT	priv_workspace_section_lookups - Private workspace section lookups

Table 172. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
PRIV_WORKSPACE_SIZE_TOP	BIGINT	priv_workspace_size_top - Maximum private workspace size
SHR_WORKSPACE_NUM_OVERFLOWS	BIGINT	shr_workspace_num_overflows - Shared workspace overflows
SHR_WORKSPACE_SECTION_INSERTS	BIGINT	shr_workspace_section_inserts - Shared workspace section inserts
SHR_WORKSPACE_SECTION_LOOKUPS	BIGINT	shr_workspace_section_lookups - Shared workspace section lookups
SHR_WORKSPACE_SIZE_TOP	BIGINT	shr_workspace_size_top - Maximum shared workspace size
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.
CAT_CACHE_SIZE_TOP	BIGINT	cat_cache_size_top - Catalog cache high water mark
TOTAL_OLAP_FUNCS	BIGINT	The total number of OLAP functions executed.
OLAP_FUNC_OVERFLOWS	BIGINT	The number of times that OLAP function data exceeded the available sort heap space.
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPBP administrative view and SNAP_GET_BP_V95 table function - Retrieve bufferpool logical group snapshot information

The SNAPBP administrative view and the SNAP_GET_BP_V95 table function return information about buffer pools from a bufferpool snapshot, in particular, the bufferpool logical data group.

SNAPBP administrative view

This administrative view allows you to retrieve bufferpool logical group snapshot information for the currently connected database.

Used with the SNAPBP_PART administrative view, the SNAPBP administrative view provides the data equivalent to the GET SNAPSHOT FOR BUFFERPOOLS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 173 on page 599 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPBP administrative view
- CONTROL privilege on the SNAPBP administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_BP_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve data and index writes for all the bufferpools of the currently connected database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME,SUBSTR(BP_NAME,1,15)
   AS BP_NAME,POOL_DATA_WRITES,POOL_INDEX_WRITES
   FROM SYSIBMADM.SNAPBP
```

The following is an example of output from this query.

```
DB_NAME  BP_NAME         POOL_DATA_WRITES     POOL_INDEX_WRITES
-------- --------------- -------------------- --------------------
TEST     IBMDEFAULTBP                       0                    0
TEST     IBMSYSTEMBP4K                      0                    0
TEST     IBMSYSTEMBP8K                      0                    0
TEST     IBMSYSTEMBP16K                     0                    0
TEST     IBMSYSTEMBP32K                     0                    0

5 record(s) selected
```

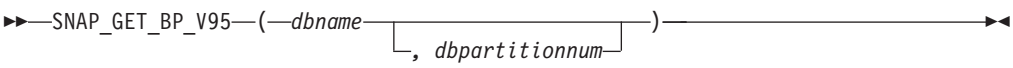
SNAP_GET_BP_V95 table function

The SNAP_GET_BP_V95 table function returns the same information as the SNAPBP administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_BP_PART table function, the SNAP_GET_BP_V95 table function provides the data equivalent to the GET SNAPSHOT FOR ALL BUFFERPOOLS CLP command.

Refer to Table 173 on page 599 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_BP_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_BP_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve total physical and logical reads for all bufferpools for all active databases for the currently connected database partition.

```
SELECT SUBSTR(T.DB_NAME,1,10) AS DB_NAME,
   SUBSTR(T.BP_NAME,1,20) AS BP_NAME,
   (T.POOL_DATA_L_READS+T.POOL_INDEX_L_READS) AS TOTAL_LOGICAL_READS,
```

```
(T.POOL_DATA_P_READS+T.POOL_INDEX_P_READS) AS TOTAL_PHYSICAL_READS,
T.DBPARTITIONNUM
FROM TABLE(SNAP_GET_BP_V95(CAST(NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
DB_NAME    BP_NAME          TOTAL_LOGICAL_READS  ...
---------- ------------...- -------------------- ...
SAMPLE     IBMDEFAULTBP                        0 ...
TOOLSDB    IBMDEFAULTBP                        0 ...
TOOLSDB    BP32K0000                           0 ...

  3 record(s) selected.
```

Output from this query (continued).

```
... TOTAL_PHYSICAL_READS DBPARTITIONNUM
... -------------------- --------------
...                    0              0
...                    0              0
...                    0              0
```

Information returned

Table 173. Information returned by the SNAPBP administrative view and the SNAP_GET_BP_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time

Table 173. Information returned by the SNAPBP administrative view and the SNAP_GET_BP_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
FILES_CLOSED	BIGINT	files_closed - Database files closed
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads

Table 173. Information returned by the SNAPBP administrative view and the SNAP_GET_BP_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
PAGES_FROM_BLOCK_IOS	BIGINT	pages_from_block_ios - Total number of pages read by block I/O
PAGES_FROM_VECTORED_IOS	BIGINT	pages_from_vectored_ios - Total pages read by vectored I/O
VECTORED_IOS	BIGINT	vectored_ios - Number of vectored I/O requests
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPBP_PART administrative view and SNAP_GET_BP_PART table function – Retrieve bufferpool_nodeinfo logical data group snapshot information

The SNAPBP_PART administrative view and the SNAP_GET_BP_PART table function return information about buffer pools from a bufferpool snapshot, in particular, the bufferpool_nodeinfo logical data group.

SNAPBP_PART administrative view

This administrative view allows you to retrieve bufferpool_nodeinfo logical data group snapshot information for the currently connected database.

Used with the SNAPBP administrative view, the SNAPBP_PART administrative view provides the data equivalent to the GET SNAPSHOT FOR BUFFERPOOLS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 174 on page 604 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:
- SELECT privilege on the SNAPBP_PART administrative view
- CONTROL privilege on the SNAPBP_PART administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the SNAP_GET_BP_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve data for all bufferpools when connected to SAMPLE database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(BP_NAME,1,15) AS BP_NAME,
   BP_CUR_BUFFSZ, BP_NEW_BUFFSZ, BP_PAGES_LEFT_TO_REMOVE, BP_TBSP_USE_COUNT
   FROM SYSIBMADM.SNAPBP_PART
```

The following is an example of output from this query.

```
DB_NAME  BP_NAME         BP_CUR_BUFFSZ        BP_NEW_BUFFSZ        ...
-------- --------------- -------------------- -------------------- ...
SAMPLE   IBMDEFAULTBP                   1000                 1000 ...
SAMPLE   IBMSYSTEMBP4K                    16                   16 ...
SAMPLE   IBMSYSTEMBP8K                    16                   16 ...
SAMPLE   IBMSYSTEMBP16K                   16                   16 ...
                                                                  ...
  4 record(s) selected.
```

Output from this query (continued).

```
... BP_PAGES_LEFT_TO_REMOVE BP_TBSP_USE_COUNT
... ----------------------- --------------------
...                       0                    3
...                       0                    0
...                       0                    0
...                       0                    0
...
```

SNAP_GET_BP_PART table function

The SNAP_GET_BP_PART table function returns the same information as the SNAPBP_PART administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_BP_V95 table function, the SNAP_GET_BP_PART table function provides the data equivalent to the GET SNAPSHOT FOR ALL BUFFERPOOLS CLP command.

Refer to Table 174 on page 604 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot for all bufferpools in all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_BP_PART table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_BP_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve data for all bufferpools for all active databases when connected to the SAMPLE database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(BP_NAME,1,15) AS BP_NAME,
   BP_CUR_BUFFSZ, BP_NEW_BUFFSZ, BP_PAGES_LEFT_TO_REMOVE, BP_TBSP_USE_COUNT
   FROM TABLE(SNAP_GET_BP_PART(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  BP_NAME         BP_CUR_BUFFSZ        BP_NEW_BUFFSZ        ...
-------- -------------- -------------------- -------------------- ...
SAMPLE   IBMDEFAULTBP                   250                  250 ...
SAMPLE   IBMSYSTEMBP4K                   16                   16 ...
SAMPLE   IBMSYSTEMBP8K                   16                   16 ...
SAMPLE   IBMSYSTEMBP16K                  16                   16 ...
SAMPLE   IBMSYSTEMBP32K                  16                   16 ...
TESTDB   IBMDEFAULTBP                   250                  250 ...
TESTDB   IBMSYSTEMBP4K                   16                   16 ...
TESTDB   IBMSYSTEMBP8K                   16                   16 ...
TESTDB   IBMSYSTEMBP16K                  16                   16 ...
TESTDB   IBMSYSTEMBP32K                  16                   16 ...

...
```

Output from this query (continued).

```
... BP_PAGES_LEFT_TO_REMOVE BP_TBSP_USE_COUNT
... ----------------------- --------------------
...                       0                    3
...                       0                    0
...                       0                    0
...                       0                    0
...                       0                    0
...                       0                    3
...                       0                    0
...                       0                    0
...                       0                    0
...                       0                    0

...
```

Information returned

Table 174. Information returned by the SNAPBP_PART administrative view and the SNAP_GET_BP_PART table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
DB_NAME	VARCHAR(128)	db_name - Database name
BP_CUR_BUFFSZ	BIGINT	bp_cur_buffsz - current size of buffer pool
BP_NEW_BUFFSZ	BIGINT	bp_new_buffsz - New buffer pool size
BP_PAGES_LEFT_TO_REMOVE	BIGINT	bp_pages_left_to_remove - Number of pages left to remove
BP_TBSP_USE_COUNT	BIGINT	bp_tbsp_use_count - Number of table spaces mapped to buffer pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPCONTAINER administrative view and SNAP_GET_CONTAINER_V91 table function - Retrieve tablespace_container logical data group snapshot information

The SNAPCONTAINER administrative view and the SNAP_GET_CONTAINER_V91 table function return table space snapshot information from the tablespace_container logical data group.

SNAPCONTAINER administrative view

This administrative view allows you to retrieve tablespace_container logical data group snapshot information for the currently connected database.

Used with the SNAPTBSP, SNAPTBSP_PART, SNAPTBSP_QUIESCER and SNAPTBSP_RANGE administrative views, the SNAPCONTAINER administrative view returns data equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 175 on page 608 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPCONTAINER administrative view
- CONTROL privilege on the SNAPCONTAINER administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_CONTAINER_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details for the table space containers for all database partitions for the currently connected database.

```
SELECT SNAPSHOT_TIMESTAMP, SUBSTR(TBSP_NAME, 1, 15) AS TBSP_NAME,
   TBSP_ID, SUBSTR(CONTAINER_NAME, 1, 20) AS CONTAINER_NAME,
   CONTAINER_ID, CONTAINER_TYPE, ACCESSIBLE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPCONTAINER ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         TBSP_NAME       TBSP_ID     ...
-------------------------- --------------- -------...- ...
2006-01-08-16.49.24.639945 SYSCATSPACE              0 ...
2006-01-08-16.49.24.639945 TEMPSPACE1               1 ...
2006-01-08-16.49.24.639945 USERSPACE1               2 ...
2006-01-08-16.49.24.639945 SYSTOOLSPACE             3 ...
2006-01-08-16.49.24.640747 TEMPSPACE1               1 ...
2006-01-08-16.49.24.640747 USERSPACE1               2 ...
2006-01-08-16.49.24.639981 TEMPSPACE1               1 ...
2006-01-08-16.49.24.639981 USERSPACE1               2 ...
                                                      ...
8 record(s) selected.
```

Output from this query (continued).

```
... CONTAINER_NAME       CONTAINER_ID     CONTAINER_TYPE   ...
... -------------------- ------------...- ---------------- ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 PATH             ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 PATH             ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 PATH             ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
```

Output from this query (continued).

```
... ACCESSIBLE DBPARTITIONNUM
... ---------- --------------
...          1              0
...          1              0
...          1              0
...          1              0
...          1              1
...          1              1
...          1              2
...          1              2
```

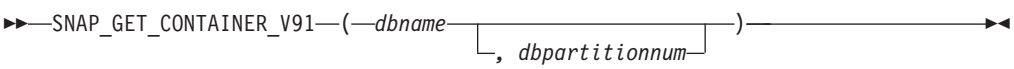
SNAP_GET_CONTAINER_V91 table function

The SNAP_GET_CONTAINER_V91 table function returns the same information as the SNAPCONTAINER administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TBSP_V91, SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_QUIESCER and SNAP_GET_TBSP_RANGE table functions, the SNAP_GET_CONTAINER_V91 table function returns data equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 175 on page 608 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_CONTAINER_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_CONTAINER_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details for the table space containers on the currently connected database on the currently connected database partition.

```
SELECT SNAPSHOT_TIMESTAMP, TBSP_NAME, TBSP_ID, CONTAINER_NAME,
   CONTAINER_ID, CONTAINER_TYPE, ACCESSIBLE
   FROM TABLE(SNAP_GET_CONTAINER_V91('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         TBSP_NAME            TBSP_ID ...
-------------------------- -------------------- ------- ...
2005-04-25-14.42.10.899253 SYSCATSPACE                0 ...
2005-04-25-14.42.10.899253 TEMPSPACE1                 1 ...
2005-04-25-14.42.10.899253 USERSPACE1                 2 ...
2005-04-25-14.42.10.899253 SYSTOOLSPACE               3 ...
2005-04-25-14.42.10.899253 MYTEMP                     4 ...
2005-04-25-14.42.10.899253 WHATSNEWTEMPSPACE          5 ...
```

Output from this query (continued).

```
... CONTAINER_NAME                              CONTAINER_ID ...
... ------------------------------------------ ------------ ...
... D:\DB2\NODE0000\SQL00002\SQLT0000.0                    0 ...
... D:\DB2\NODE0000\SQL00002\SQLT0001.0                    0 ...
... D:\DB2\NODE0000\SQL00002\SQLT0002.0                    0 ...
... D:\DB2\NODE0000\SQL00002\SYSTOOLSPACE                  0 ...
... D:\DB2\NODE0000\SQL003                                 0 ...
... d:\DGTTsWhatsNewContainer                              0 ...
```

Output from this query (continued).

```
... CONTAINER_TYPE ACCESSIBLE
... -------------- ----------
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
```

Information returned

NOTE: The BUFFERPOOL database manager monitor switch must be turned on in order for the file system information to be returned.

Table 175. Information returned by the SNAPCONTAINER administrative view and the SNAP_GET_CONTAINER_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
CONTAINER_NAME	VARCHAR(256)	container_name - Container name
CONTAINER_ID	BIGINT	container_id - Container identification
CONTAINER_TYPE	VARCHAR(16)	container_type - Container type. This is a text identifier based on the defines in `sqlutil.h` and is one of: • DISK_EXTENT_TAG • DISK_PAGE_TAG • FILE_EXTENT_TAG • FILE_PAGE_TAG • PATH
TOTAL_PAGES	BIGINT	container_total_pages - Total pages in container
USABLE_PAGES	BIGINT	container_usable_pages - Usable pages in container
ACCESSIBLE	SMALLINT	container_accessible - Accessibility of container
STRIPE_SET	BIGINT	container_stripe_set - Stripe set

Table 175. Information returned by the SNAPCONTAINER administrative view and the SNAP_GET_CONTAINER_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
FS_ID	VARCHAR(22)	fs_id - Unique file system identification number
FS_TOTAL_SIZE	BIGINT	fs_total_size - Total size of a file system
FS_USED_SIZE	BIGINT	fs_used_size - Amount of space used on a file system

SNAPDB administrative view and SNAP_GET_DB_V97 table function - Retrieve snapshot information from the dbase logical group

The SNAPDB administrative view and the SNAP_GET_DB_V97 table function return snapshot information from the database (dbase) logical group.

SNAPDB administrative view

This administrative view allows you to retrieve snapshot information from the dbase logical group for the currently connected database.

Used in conjunction with the SNAPDB_MEMORY_POOL, SNAPDETAILLOG, SNAPHADR and SNAPSTORAGE_PATHS administrative views, the SNAPDB administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE on database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 176 on page 613 for a complete list of information that is returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDB administrative view
- CONTROL privilege on the SNAPDB administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DB_V97 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve the status, platform, location, and connect time for all database partitions of the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDB ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME     DB_STATUS    SERVER_PLATFORM DB_LOCATION  ...
-------...- ------------ --------------- ------------ ...
TEST        ACTIVE       AIX64           LOCAL        ...
TEST        ACTIVE       AIX64           LOCAL        ...
TEST        ACTIVE       AIX64           LOCAL        ...

3 record(s) selected.
```

Output from this query (continued).

```
... DB_CONN_TIME               DBPARTITIONNUM
... -------------------------- --------------
... 2006-01-08-16.48.30.665477              0
... 2006-01-08-16.48.34.005328              1
... 2006-01-08-16.48.34.007937              2
```

This routine can be used by calling the following on the command line:

```
SELECT TOTAL_OLAP_FUNCS, OLAP_FUNC_OVERFLOWS, ACTIVE_OLAP_FUNCS
   FROM SYSIBMADM.SNAPDB

TOTAL_OLAP_FUNCS     OLAP_FUNC_OVERFLOWS  ACTIVE_OLAP_FUNCS
-------------------- -------------------- -----------------
                   7                    2                 1

  1 record(s) selected.
```

After running a workload, a user can use the following query:

```
SELECT STATS_CACHE_SIZE, STATS_FABRICATIONS, SYNC_RUNSTATS,
   ASYNC_RUNSTATS, STATS_FABRICATE_TIME, SYNC_RUNSTATS_TIME
   FROM SYSIBMADM.SNAPDB

STATS_CACHE_SIZE STATS_FABRICATIONS SYNC_RUNSTATS ASYNC_RUNSTATS ...
---------------- ----------------- ------------- -------------- ...
             128                 2             1              0 ...


... STATS_FABRICATE_TIME SYNC_RUNSTATS_TIME
... -------------------- ------------------
...                   10                100

1 record(s) selected.
```

SNAP_GET_DB_V97 table function

The SNAP_GET_DB_V97 table function returns the same information as the SNAPDB administrative view.

Note: If your database was created in Version 9.7 prior to Fix Pack 1, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7 , it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database migration). If you downgrade to Version 9.7 , this routine will no longer work.

Used in conjunction with the SNAP_GET_DB_MEMORY_POOL, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DB_V97 table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 176 on page 613 for a complete list of information that is returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DB_V97 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DB_V97 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Example 1: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions of the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME FROM TABLE(SNAP_GET_DB_V97('', -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME     DB_STATUS    SERVER_PLATFORM ...
-------...- ---------... -------------- ...
SAMPLE      ACTIVE       AIX64          ...

  1 record(s) selected.
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.09.22.013196
```

Example 2: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions for all active databases in the same instance that contains the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME
   FROM TABLE(SNAP_GET_DB_V97(CAST (NULL AS VARCHAR(128)), -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME      DB_STATUS    SERVER_PLATFORM ...
--------...- ---------... -------------- ...
TOOLSDB      ACTIVE       AIX64          ...
SAMPLE       ACTIVE       AIX64          ...
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.26.54.396335
... LOCAL        2005-07-24-22.09.22.013196
```

Example 3: This routine can be used by calling the following on the command line:

When connected to a database:

```
SELECT TOTAL_OLAP_FUNCS, OLAP_FUNC_OVERFLOWS, ACTIVE_OLAP_FUNCS
   FROM TABLE (SNAP_GET_DB_V97('', 0)) AS T

The output will look like:

TOTAL_OLAP_FUNCS  OLAP_FUNC_OVERFLOWS  ACTIVE_OLAP_FUNCS
----------------  -------------------- --------------------
               7                    2                    1

1 record(s) selected.
```

Example 4: After running a workload, a user can use the following query with the table function.

```
 SELECT STATS_CACHE_SIZE, STATS_FABRICATIONS, SYNC_RUNSTATS,
   ASYNC_RUNSTATS, STATS_FABRICATE_TIME, SYNC_RUNSTATS_TIME
   FROM TABLE (SNAP_GET_DB_V97('mytestdb', -1)) AS SNAPDB

STATS_CACHE_SIZE STATS_FABRICATIONS SYNC_RUNSTATS ASYNC_RUNSTATS ...
```

```
--------------- ----------------- ------------ ------------- ...
            200                 1            2             0 ...

Continued

...STATS_FABRICATE_TIME  SYNC_RUNSTATS_TIME
...----------------------  ------------------
...                    2                  32

  1 record(s) selected.
```

Example 5: The following example shows how you can use the SNAP_GET_DB_V97 table function to determine the status of a database:

```
SELECT SUBSTR
   (DB_NAME, 1, 20) AS DB_NAME, DB_STATUS
    FROM  table(SNAP_GET_DB_V97('hadrdb', 0))

DB_NAME              DB_STATUS
-------------------- ---------------
HADRDB               ACTIVE_STANDBY
```

SNAPDB administrative view and SNAP_GET_DB_V97 table function metadata

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
DB_STATUS	VARCHAR(16)	db_status - Status of database. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ACTIVE • QUIESCE_PEND • QUIESCED • ROLLFWD • ACTIVE_STANDBY - the HADR database is in a standby mode with reads on standby enabled. • STANDBY - the HADR database is in standby mode (reads on standby are not enabled).
CATALOG_PARTITION	SMALLINT	catalog_node - Catalog node number
CATALOG_PARTITION_NAME	VARCHAR(128)	catalog_node_name - Catalog node network name

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
SERVER_PLATFORM	VARCHAR(12)	server_platform - Server operating system. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS
DB_LOCATION	VARCHAR(12)	db_location - Database location. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • LOCAL • REMOTE
DB_CONN_TIME	TIMESTAMP	db_conn_time - Database activation timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
LAST_BACKUP	TIMESTAMP	last_backup - Last backup timestamp

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
CONNECTIONS_TOP	BIGINT	connections_top - Maximum number of concurrent connections
TOTAL_CONS	BIGINT	total_cons - Connects since database activation
TOTAL_SEC_CONS	BIGINT	total_sec_cons - Secondary connections
APPLS_CUR_CONS	BIGINT	appls_cur_cons - Applications connected currently
APPLS_IN_DB2	BIGINT	appls_in_db2 - Applications executing in the database currently
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
AGENTS_TOP	BIGINT	agents_top - Number of agents created
COORD_AGENTS_TOP	BIGINT	coord_agents_top - Maximum number of coordinating agents
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_LIST_IN_USE	BIGINT	lock_list_in_use - Total lock list memory in use
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
LOCKS_WAITING	BIGINT	locks_waiting - Current agents waiting on locks
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
NUM_INDOUBT_TRANS	BIGINT	num_indoubt_trans - Number of indoubt transactions
SORT_HEAP_ALLOCATED	BIGINT	sort_heap_allocated - Total sort heap allocated
SORT_SHRHEAP_ALLOCATED	BIGINT	sort_shrheap_allocated - Sort share heap currently allocated
SORT_SHRHEAP_TOP	BIGINT	sort_shrheap_top - Sort share heap high water mark
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
TOTAL_SORTS	BIGINT	total_sorts - Total sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
ACTIVE_SORTS	BIGINT	active_sorts - Active sorts
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
POOL_LSN_GAP_CLNS	BIGINT	pool_lsn_gap_clns - Buffer pool log space cleaners triggered
POOL_DRTY_PG_STEAL_CLNS	BIGINT	pool_drty_pg_steal_clns - Buffer pool victim page cleaners triggered
POOL_DRTY_PG_THRSH_CLNS	BIGINT	pool_drty_pg_thrsh_clns - Buffer pool threshold cleaners triggered
PREFETCH_WAIT_TIME	BIGINT	prefetch_wait_time - Time waited for prefetch
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
FILES_CLOSED	BIGINT	files_closed - Database files closed
ELAPSED_EXEC_TIME_S	BIGINT	elapsed_exec_time - Statement execution elapsed time
ELAPSED_EXEC_TIME_MS	BIGINT	elapsed_exec_time - Statement execution elapsed time
COMMIT_SQL_STMTS	BIGINT	commit_sql_stmts - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	rollback_sql_stmts - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	dynamic_sql_stmts - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	static_sql_stmts - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	failed_sql_stmts - Failed statement operations
SELECT_SQL_STMTS	BIGINT	select_sql_stmts - Select SQL statements executed

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
UID_SQL_STMTS	BIGINT	uid_sql_stmts - UPDATE/INSERT/DELETE SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements
INT_AUTO_REBINDS	BIGINT	int_auto_rebinds - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
INT_COMMITS	BIGINT	int_commits - Internal commits
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	int_deadlock_rollbacks - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_SELECTED	BIGINT	rows_selected - Rows selected
ROWS_READ	BIGINT	rows_read - Rows read
BINDS_PRECOMPILES	BIGINT	binds_precompiles - Binds/precompiles attempted
TOTAL_LOG_AVAILABLE	BIGINT	total_log_available - Total log available
TOTAL_LOG_USED	BIGINT	total_log_used - Total log space used
SEC_LOG_USED_TOP	BIGINT	sec_log_used_top - Maximum secondary log space used
TOT_LOG_USED_TOP	BIGINT	tot_log_used_top - Maximum total log space used
SEC_LOGS_ALLOCATED	BIGINT	sec_logs_allocated - Secondary logs allocated currently
LOG_READS	BIGINT	log_reads - Number of log pages read
LOG_READ_TIME_S	BIGINT	log_read_time - Log read time
LOG_READ_TIME_NS	BIGINT	log_read_time - Log read time
LOG_WRITES	BIGINT	log_writes - Number of log pages written
LOG_WRITE_TIME_S	BIGINT	log_write_time - Log write time
LOG_WRITE_TIME_NS	BIGINT	log_write_time - Log write time
NUM_LOG_WRITE_IO	BIGINT	num_log_write_io - Number of log writes
NUM_LOG_READ_IO	BIGINT	num_log_read_io - Number of log reads
NUM_LOG_PART_PAGE_IO	BIGINT	num_log_part_page_io - Number of partial log page writes
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers
NUM_LOG_DATA_FOUND_ IN_BUFFER	BIGINT	num_log_data_found_in_buffer - Number of log data found in buffer

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
APPL_ID_OLDEST_XACT	BIGINT	appl_id_oldest_xact - Application with oldest transaction
LOG_TO_REDO_FOR_ RECOVERY	BIGINT	log_to_redo_for_recovery - Amount of log to be redone for recovery
LOG_HELD_BY_DIRTY_PAGES	BIGINT	log_held_by_dirty_pages - Amount of log space accounted for by dirty pages
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
PKG_CACHE_NUM_ OVERFLOWS	BIGINT	pkg_cache_num_overflows - Package cache overflows
PKG_CACHE_SIZE_TOP	BIGINT	pkg_cache_size_top - Package cache high water mark
APPL_SECTION_LOOKUPS	BIGINT	appl_section_lookups - Section lookups
APPL_SECTION_INSERTS	BIGINT	appl_section_inserts - Section inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	cat_cache_overflows - Catalog cache overflows
CAT_CACHE_SIZE_TOP	BIGINT	cat_cache_size_top - Catalog cache high water mark
PRIV_WORKSPACE_SIZE_TOP	BIGINT	priv_workspace_size_top - Maximum private workspace size
PRIV_WORKSPACE_NUM_ OVERFLOWS	BIGINT	priv_workspace_num_overflows - Private workspace overflows
PRIV_WORKSPACE_SECTION_ INSERTS	BIGINT	priv_workspace_section_inserts - Private workspace section inserts
PRIV_WORKSPACE_SECTION_ LOOKUPS	BIGINT	priv_workspace_section_lookups - Private workspace section lookups
SHR_WORKSPACE_SIZE_TOP	BIGINT	shr_workspace_size_top - Maximum shared workspace size
SHR_WORKSPACE_NUM_ OVERFLOWS	BIGINT	shr_workspace_num_overflows - Shared workspace overflows
SHR_WORKSPACE_SECTION_ INSERTS	BIGINT	shr_workspace_section_inserts - Shared workspace section inserts
SHR_WORKSPACE_SECTION_ LOOKUPS	BIGINT	shr_workspace_section_lookups - Shared workspace section lookups
TOTAL_HASH_JOINS	BIGINT	total_hash_joins - Total hash joins

Table 176. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	hash_join_overflows - Hash join overflows
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	hash_join_small_overflows - Hash join small overflows
POST_SHRTHRESHOLD_ HASH_JOINS	BIGINT	post_shrthreshold_hash_joins - Post threshold hash joins
ACTIVE_HASH_JOINS	BIGINT	active_hash_joins - Active hash joins
NUM_DB_STORAGE_PATHS	BIGINT	num_db_storage_paths - Number of automatic storage paths
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
SMALLEST_LOG_AVAIL_ NODE	INTEGER	smallest_log_avail_node - Node with least available log space
TOTAL_OLAP_FUNCS	BIGINT	The total number of OLAP functions executed.
OLAP_FUNC_OVERFLOWS	BIGINT	The number of times that OLAP function data exceeded the available sort heap space.
ACTIVE_OLAP_FUNCS	BIGINT	The total number of OLAP functions that are currently running and consuming sort heap memory.
STATS_CACHE_SIZE	BIGINT	The size of the statistics cache in bytes
STATS_FABRICATIONS	BIGINT	Total number of statistics-collect activities for creating statistics by the system without table or index scan.
SYNC_RUNSTATS	BIGINT	Total number of synchronous statistics-collect activities during query compilation.
ASYNC_RUNSTATS	BIGINT	We will change the output for this column to total number of successful asynchronous statistics-collect activities.
STATS_FABRICATE_TIME	BIGINT	Total time spent on creating statistics by system without table or index scan during query compilation in milliseconds.
SYNC_RUNSTATS_TIME	BIGINT	Total time spent on synchronous statistics-collect activities in milliseconds.
NUM_THRESHOLD_VIOLATIONS	BIGINT	The number of threshold violations that have occurred at the database.

SNAPDB_MEMORY_POOL administrative view and SNAP_GET_DB_MEMORY_POOL table function – Retrieve database level memory usage information

The SNAPDB_MEMORY_POOL administrative view and the SNAP_GET_DB_MEMORY_POOL table function return information about memory usage at the database level for UNIX platforms only.

SNAPDB_MEMORY_POOL administrative view

This administrative view allows you to retrieve database level memory usage information for the currently connected database.

Used with the SNAPDB, SNAPDETAILLOG, SNAPHADR and SNAPSTORAGE_PATHS administrative views, the SNAPDB_MEMORY_POOL administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 177 on page 624 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDB_MEMORY_POOL administrative view
- CONTROL privilege on the SNAPDB_MEMORY_POOL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DB_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size for the currently connected database, SAMPLE.

```
SELECT POOL_ID, POOL_CUR_SIZE FROM SYSIBMADM.SNAPDB_MEMORY_POOL
```

The following is an example of output from this query.

```
POOL_ID       POOL_CUR_SIZE
------------- --------------------
UTILITY                      32768
PACKAGE_CACHE               475136
CAT_CACHE                    65536
```

```
BP                          2097152
BP                          1081344
BP                           540672
BP                           278528
BP                           147456
BP                            81920
LOCK_MGR                     294912
DATABASE                    3833856
OTHER                             0

  12 record(s) selected.
```

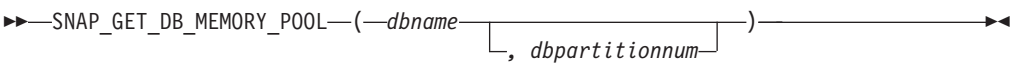
SNAP_GET_DB_MEMORY_POOL table function

The SNAP_GET_DB_MEMORY_POOL table function returns the same information as the SNAPDB_MEMORY_POOL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DB_V95, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DB_MEMORY_POOL table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 177 on page 624 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the

SNAP_GET_DB_MEMORY_POOL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DB_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size for all databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, POOL_ID, POOL_CUR_SIZE
   FROM TABLE(SNAPSHOT_GET_DB_MEMORY_POOL
   (CAST(NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  POOL_ID        POOL_CUR_SIZE
-------- -------------- --------------------
TESTDB   UTILITY                  65536
TESTDB   PACKAGE_CACHE           851968
TESTDB   CAT_CACHE                65536
TESTDB   BP                    35913728
TESTDB   BP                      589824
TESTDB   BP                      327680
TESTDB   BP                      196608
TESTDB   BP                      131072
TESTDB   SHARED_SORT              65536
TESTDB   LOCK_MGR                10092544
TESTDB   DATABASE                4980736
TESTDB   OTHER                    196608
SAMPLE   UTILITY                  65536
SAMPLE   PACKAGE_CACHE           655360
SAMPLE   CAT_CACHE               131072
SAMPLE   BP                     4325376
SAMPLE   BP                      589824
SAMPLE   BP                      327680
SAMPLE   BP                      196608
SAMPLE   BP                      131072
SAMPLE   SHARED_SORT                  0
SAMPLE   LOCK_MGR                655360
SAMPLE   DATABASE                4653056
SAMPLE   OTHER                    196608

24 record(s) selected.
```

Information returned

Table 177. Information returned by the SNAPDB_MEMORY_POOL administrative view and the SNAP_GET_DB_MEMORY_POOL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
POOL_ID	VARCHAR(14)	pool_id - Memory pool identifier. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • APP_GROUP • APPL_CONTROL • APPLICATION • BP • CAT_CACHE • DATABASE • DFM • FCMBP • IMPORT_POOL • LOCK_MGR • MONITOR • OTHER • PACKAGE_CACHE • QUERY • SHARED_SORT • SORT • STATEMENT • STATISTICS • UTILITY
POOL_SECONDARY_ID	VARCHAR(32)	pool_secondary_id - Memory pool secondary identifier
POOL_CUR_SIZE	BIGINT	pool_cur_size - Current size of memory pool
POOL_WATERMARK	BIGINT	pool_watermark - Memory pool watermark
POOL_CONFIG_SIZE	BIGINT	pool_config_size - Configured size of memory pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPDBM administrative view and SNAP_GET_DBM_V95 table function - Retrieve the dbm logical grouping snapshot information

The SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function return the snapshot monitor DB2 database manager (dbm) logical grouping information.

SNAPDBM administrative view

Used with the SNAPDBM_MEMORY_POOL, SNAPFCM, SNAPFCM_PART and SNAPSWITCHES administrative views, the SNAPDBM administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 178 on page 627 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDBM administrative view
- CONTROL privilege on the SNAPDBM administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DBM_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve database manager status and connection information for all database partitions.

```
SELECT DB2_STATUS, DB2START_TIME, LAST_RESET, LOCAL_CONS, REM_CONS_IN,
   (AGENTS_CREATED_EMPTY_POOL/AGENTS_FROM_POOL) AS AGENT_USAGE,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPDBM ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB2_STATUS   DB2START_TIME             LAST_RESET     ...
------------ -------------------------- ----------...- ...
ACTIVE       2006-01-06-14.59.59.059879              - ...
ACTIVE       2006-01-06-14.59.59.097605              - ...
ACTIVE       2006-01-06-14.59.59.062798              - ...

  3 record(s) selected.                                ...
```

Output from this query (continued).

```
... LOCAL_CONS     REM_CONS_IN     AGENT_USAGE     DBPARTITIONNUM
... ----------...- -----------...- -----------...- --------------
...              1               1               0              0
...              0               0               0              1
...              0               0               0              2
```

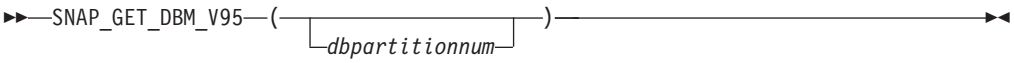
SNAP_GET_DBM_V95 table function

The SNAP_GET_DBM_V95 table function returns the same information as the SNAPDBM administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_MEMORY_POOL, SNAP_GET_FCM, SNAP_GET_FCM_PART and SNAP_GET_SWITCHES table functions, the SNAP_GET_DBM_V95 table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 178 on page 627 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DBM_V95 table function calls the snapshot from memory.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DBM_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the start time and current status of database partition number 2.

```
SELECT DB2START_TIME, DB2_STATUS FROM TABLE(SNAP_GET_DBM_V95(2)) AS T
```

The following is an example of output from this query.

```
DB2START_TIME              DB2_STATUS
-------------------------- ------------
2006-01-06-14.59.59.062798 ACTIVE
```

Information returned

Table 178. Information returned by the SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
SORT_HEAP_ALLOCATED	BIGINT	sort_heap_allocated - Total sort heap allocated
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
PIPED_SORTS_REQUESTED	BIGINT	piped_sorts_requested - Piped sorts requested
PIPED_SORTS_ACCEPTED	BIGINT	piped_sorts_accepted - Piped sorts accepted
REM_CONS_IN	BIGINT	rem_cons_in - Remote connections to database manager
REM_CONS_IN_EXEC	BIGINT	rem_cons_in_exec - Remote Connections Executing in the Database Manager monitor element
LOCAL_CONS	BIGINT	local_cons - Local connections
LOCAL_CONS_IN_EXEC	BIGINT	local_cons_in_exec - Local Connections Executing in the Database Manager monitor element
CON_LOCAL_DBASES	BIGINT	con_local_dbases - Local databases with current connects
AGENTS_REGISTERED	BIGINT	agents_registered - Agents registered
AGENTS_WAITING_ON_TOKEN	BIGINT	agents_waiting_on_token - Agents waiting for a token
DB2_STATUS	VARCHAR(12)	db2_status - Status of DB2 instance This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ACTIVE • QUIESCE_PEND • QUIESCED
AGENTS_REGISTERED_TOP	BIGINT	agents_registered_top - Maximum number of agents registered
AGENTS_WAITING_TOP	BIGINT	agents_waiting_top - Maximum number of agents waiting
COMM_PRIVATE_MEM	BIGINT	comm_private_mem - Committed private memory
IDLE_AGENTS	BIGINT	idle_agents - Number of idle agents

Table 178. Information returned by the SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
AGENTS_FROM_POOL	BIGINT	agents_from_pool - Agents assigned from pool
AGENTS_CREATED_EMPTY_POOL	BIGINT	agents_created_empty_pool - Agents created due to empty agent pool
COORD_AGENTS_TOP	BIGINT	coord_agents_top - Maximum number of coordinating agents
MAX_AGENT_OVERFLOWS	BIGINT	max_agent_overflows - Maximum agent overflows
AGENTS_STOLEN	BIGINT	agents_stolen - Stolen agents
GW_TOTAL_CONS	BIGINT	gw_total_cons - Total number of attempted connections for DB2 Connect
GW_CUR_CONS	BIGINT	gw_cur_cons - Current number of connections for DB2 Connect
GW_CONS_WAIT_HOST	BIGINT	gw_cons_wait_host - Number of connections waiting for the host to reply
GW_CONS_WAIT_CLIENT	BIGINT	gw_cons_wait_client - Number of connections waiting for the client to send request
POST_THRESHOLD_ HASH_JOINS	BIGINT	post_threshold_hash_joins - Hash join threshold
NUM_GW_CONN_SWITCHES	BIGINT	num_gw_conn_switches - Connection switches
DB2START_TIME	TIMESTAMP	db2start_time - Start database manager timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
NUM_NODES_IN_ DB2_INSTANCE	INTEGER	num_nodes_in_db2_instance - Number of nodes in database partition
PRODUCT_NAME	VARCHAR(32)	product_name - Product name
SERVICE_LEVEL	VARCHAR(18)	service_level - Service level
SORT_HEAP_TOP	BIGINT	sort_heap_top - Sort private heap high water mark
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

Table 178. Information returned by the SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POST_THRESHOLD_OLAP_FUNCS	BIGINT	The number of OLAP functions which have requested a sort heap after the sort heap threshold has been exceeded. Sorts, hash joins, and OLAP functions are examples of operations which utilize a sort heap. Under normal conditions, the database manager will allocate sort heap using the value specified by the `sortheap` configuration parameter. If the amount of memory allocated to sort heaps exceeds the sort heap threshold (`sheapthres` configuration parameter), the database manager will allocate subsequent sort heaps using a value less than that specified by the `sortheap` configuration parameter. OLAP functions which start after the sort heap threshold has been reached may not receive an optimum amount of memory to execute.

SNAPDBM_MEMORY_POOL administrative view and SNAP_GET_DBM_MEMORY_POOL table function – Retrieve database manager level memory usage information

The SNAPDBM_MEMORY_POOL administrative view and the SNAP_GET_DBM_MEMORY_POOL table function return information about memory usage at the database manager.

SNAPDBM_MEMORY_POOL administrative view

Used with the SNAPDBM, SNAPFCM, SNAPFCM_PART and SNAPSWITCHES administrative views, the SNAPDBM_MEMORY_POOL administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 179 on page 631 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDBM_MEMORY_POOL administrative view
- CONTROL privilege on the SNAPDBM_MEMORY_POOL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DBM_MEMORY_POOL table function

- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of the memory pools and their current size for the database manager of the connected database.

```
SELECT POOL_ID, POOL_CUR_SIZE FROM SNAPDBM_MEMORY_POOL
```

The following is an example of output from this query.

```
POOL_ID        POOL_CUR_SIZE
-------------- --------------------
MONITOR                     65536
OTHER                    29622272
FCMBP                    57606144
...
```

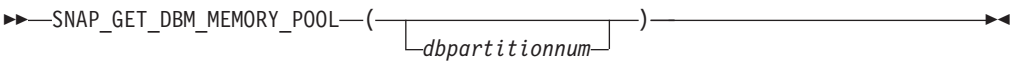
SNAP_GET_DBM_MEMORY_POOL table function

The SNAP_GET_DBM_MEMORY_POOL table function returns the same information as the SNAPDBM_MEMORY_POOL administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_V95, SNAP_GET_FCM, SNAP_GET_FCM_PART and SNAP_GET_SWITCHES table functions, the SNAP_GET_DBM_MEMORY_POOL table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 179 on page 631 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DBM_MEMORY_POOL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DBM_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of the memory pools and their current size for all database partitions of the database manager of the connected database.

```
SELECT POOL_ID, POOL_CUR_SIZE, DBPARTITIONNUM
   FROM TABLE(SYSPROC.SNAP_GET_DBM_MEMORY_POOL())
   AS T ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
POOL_ID        POOL_CUR_SIZE        DBPARTITIONNUM
-------------- -------------------- --------------
MONITOR                   65536              0
OTHER                  29622272              0
FCMBP                  57606144              0
MONITOR                   65536              1
OTHER                  29425664              1
FCMBP                  57606144              1
MONITOR                   65536              2
OTHER                  29425664              2
FCMBP                  57606144              2
```

Information returned

Table 179. Information returned by the SNAPDBM_MEMORY_POOL administrative view and the SNAP_GET_DBM_MEMORY_POOL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.

Table 179. Information returned by the SNAPDBM_MEMORY_POOL administrative view and the SNAP_GET_DBM_MEMORY_POOL table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ID	VARCHAR(14)	pool_id - Memory pool identifier. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • APP_GROUP • APPL_CONTROL • APPLICATION • BP • CAT_CACHE • DATABASE • DFM • FCMBP • IMPORT_POOL • LOCK_MGR • MONITOR • OTHER • PACKAGE_CACHE • QUERY • SHARED_SORT • SORT • STATEMENT • STATISTICS • UTILITY
POOL_CUR_SIZE	BIGINT	pool_cur_size - Current size of memory pool
POOL_WATERMARK	BIGINT	pool_watermark - Memory pool watermark
POOL_CONFIG_SIZE	BIGINT	pool_config_size - Configured size of memory pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPDETAILLOG administrative view and SNAP_GET_DETAILLOG_V91 table function - Retrieve snapshot information from the detail_log logical data group

The SNAPDETAILLOG administrative view and the SNAP_GET_DETAILLOG_V91 table function return snapshot information from the detail_log logical data group.

SNAPDETAILLOG administrative view

This administrative view allows you to retrieve snapshot information from the detail_log logical data group for the currently connected database.

Used in conjunction with the SNAPDB, SNAPDB_MEMORY_POOL, SNAPHADR and SNAPSTORAGE_PATHS administrative views, the SNAPDETAILLOG administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE on database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 180 on page 635 for a complete list of information that is returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDETAILLOG administrative view
- CONTROL privilege on the SNAPDETAILLOG administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DETAILLOG_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve log information for all database partitions for the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DB_NAME, FIRST_ACTIVE_LOG,
   LAST_ACTIVE_LOG, CURRENT_ACTIVE_LOG, CURRENT_ARCHIVE_LOG,
   DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDETAILLOG ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME  FIRST_ACTIVE_LOG     LAST_ACTIVE_LOG      ...
-------- -------------------- -------------------- ...
TEST                        0                    8 ...
TEST                        0                    8 ...
TEST                        0                    8 ...

  3 record(s) selected.
```

Output from this query (continued).

```
... CURRENT_ACTIVE_LOG   CURRENT_ARCHIVE_LOG  DBPARTITIONNUM
... -------------------- -------------------- --------------
...                    0                    -              0
...                    0                    -              1
...                    0                    -              2
```

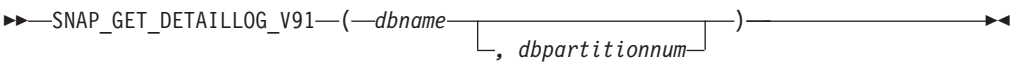
SNAP_GET_DETAILLOG_V91 table function

The SNAP_GET_DETAILLOG_V91 table function returns the same information as the SNAPDETAILLOG administrative view.

Used in conjunction with the SNAP_GET_DB_V95, SNAP_GET_DB_MEMORY_POOL, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DETAILLOG table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 180 on page 635 for a complete list of information that is returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DETAILLOG_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DETAILLOG_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve log information for database partition 1 for the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DB_NAME, FIRST_ACTIVE_LOG,
   LAST_ACTIVE_LOG, CURRENT_ACTIVE_LOG, CURRENT_ARCHIVE_LOG
   FROM TABLE(SNAP_GET_DETAILLOG_V91('', 1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  FIRST_ACTIVE_LOG     LAST_ACTIVE_LOG      ...
-------- -------------------- -------------------- ...
TEST                        0                    8 ...

  1 record(s) selected.
```

Output from this query (continued).

```
... CURRENT_ACTIVE_LOG   CURRENT_ARCHIVE_LOG
... -------------------- --------------------
...                    0                    -
```

SNAPDETAILLOG administrative view and SNAP_GET_DETAILLOG_V91 table function metadata

Table 180. Information returned by the SNAPDETAILLOG administrative view and SNAP_GET_DETAILLOG_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
FIRST_ACTIVE_LOG	BIGINT	first_active_log - First active log file number
LAST_ACTIVE_LOG	BIGINT	last_active_log - Last active log file number
CURRENT_ACTIVE_LOG	BIGINT	current_active_log - Current active log file number
CURRENT_ARCHIVE_LOG	BIGINT	current_archive_log - Current archive log file number
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPDYN_SQL administrative view and SNAP_GET_DYN_SQL_V95 table function - Retrieve dynsql logical group snapshot information

The "SNAPDYN_SQL administrative view" on page 636 and the "SNAP_GET_DYN_SQL_V95 table function" on page 637 return snapshot information from the dynsql logical data group.

SNAPDYN_SQL administrative view

This administrative view allows you to retrieve dynsql logical group snapshot information for the currently connected database.

This view returns information equivalent to the GET SNAPSHOT FOR DYNAMIC SQL ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 181 on page 639 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDYN_SQL administrative view
- CONTROL privilege on the SNAPDYN_SQL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DYN_SQL_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of dynamic SQL run on all database partitions of the currently connected database, ordered by the number of rows read.

```
SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT, 1, 60)
   AS STMT_TEXT, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDYN_SQL ORDER BY ROWS_READ
```

The following is an example of output from this query.

```
PREP_TIME_WORST      NUM_COMPILATIONS     ...
-------------------- -------------------- ...
                  98                    1 ...
                   9                    1 ...
                   0                    0 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                  40                    1 ...

  9 record(s) selected.
```

Output from this query (continued).

```
... STMT_TEXT                                                        ...
... ---------------------------------------------------------------- ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
... select * from dbuser.employee                                    ...
... SET CURRENT LOCALE LC_CTYPE = 'en_US'                            ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
... select * from dbuser.employee                                    ...
... insert into dbuser.employee values(1)                            ...
... select * from dbuser.employee                                    ...
... insert into dbuser.employee values(1)                            ...
```

Output from this query (continued).

```
... DBPARTITIONNUM
... --------------
...               0
...               0
...               0
...               2
...               1
...               2
...               2
...               1
...               0
```

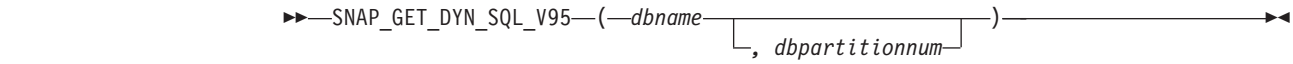
SNAP_GET_DYN_SQL_V95 table function

The SNAP_GET_DYN_SQL_V95 table function returns the same information as the SNAPDYN_SQL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

This table function returns information equivalent to the GET SNAPSHOT FOR DYNAMIC SQL ON database-alias CLP command.

Refer to Table 181 on page 639 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input

option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DYN_SQL_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DYN_SQL_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of dynamic SQL run on the currently connected database partition of the currently connected database, ordered by the number of rows read.

```
SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT, 1, 60)
   AS STMT_TEXT FROM TABLE(SNAP_GET_DYN_SQL_V95('',-1)) as T
    ORDER BY ROWS_READ
```

The following is an example of output from this query.

```
PREP_TIME_WORST      ...
-------------------- ...
                   0 ...
                   3 ...
                     ...
                   4 ...
                     ...
                   4 ...
                     ...
                   4 ...
                     ...
                   3 ...
                     ...
                   4 ...
                     ...
```

Output from this query (continued).

```
... NUM_COMPILATIONS     STMT_TEXT
... -------------------- ----------------------------------------...-
...                    0 SET CURRENT LOCALE LC_CTYPE = 'en_US'
...                    1 select rows_read, rows_written,
...                          substr(stmt_text, 1, 40) as
...                    1 select * from table
...                          (snap_get_dyn_sqlv9('',-1)) as t
...                    1 select * from table
```

```
...                         (snap_getdetaillog9('',-1)) as t
...                     1 select * from table
...                         (snap_get_hadr('',-1)) as t
...                     1 select prep_time_worst, num_compilations,
...                         substr(stmt_text,
...                     1 select prep_time_worst, num_compilations,
...                         substr(stmt_text,
```

After running a workload, user can use the following query with the table function.

```
SELECT STATS_FABRICATE_TIME,SYNC_RUNSTATS_TIME
   FROM TABLE (SNAP_GET_DYN_SQL_V95('mytestdb', -1))
   AS SNAPDB

STATS_FABRICATE_TIME   SYNC_RUNSTATS_TIME
---------------------- ------------------
                     2                  12
                     1                  30
```

For the view based on this table function:

```
SELECT STATS_FABRICATE_TIME,SYNC_RUNSTATS_TIME
   FROM SYSIBMADM.SNAPDYN_SQL

STATS_FABRICATE_TIME   SYNC_RUNSTATS_TIME
---------------------- ------------------
                     5                  10
                     3                  20
  2 record(s) selected.
```

Information returned

Table 181. Information returned by the SNAPDYN_SQL administrative view and the SNAP_GET_DYN_SQL_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
NUM_EXECUTIONS	BIGINT	num_executions - Statement executions
NUM_COMPILATIONS	BIGINT	num_compilations - Statement compilations
PREP_TIME_WORST	BIGINT	prep_time_worst - Statement worst preparation time
PREP_TIME_BEST	BIGINT	prep_time_best - Statement best preparation time
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
STMT_SORTS	BIGINT	stmt_sorts - Statement sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads

Table 181. Information returned by the SNAPDYN_SQL administrative view and the SNAP_GET_DYN_SQL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
TOTAL_EXEC_TIME	BIGINT	total_exec_time - Elapsed statement execution time (in seconds)*
TOTAL_EXEC_TIME_MS	BIGINT	total_exec_time - Elapsed statement execution time (fractional, in microseconds)*
TOTAL_USR_CPU_TIME	BIGINT	total_usr_cpu_time - Total user CPU for a statement (in seconds)*
TOTAL_USR_CPU_TIME_MS	BIGINT	total_usr_cpu_time - Total user CPU for a statement (fractional, in microseconds)*
TOTAL_SYS_CPU_TIME	BIGINT	total_sys_cpu_time - Total system CPU for a statement (in seconds)*
TOTAL_SYS_CPU_TIME_MS	BIGINT	total_sys_cpu_time - Total system CPU for a statement (fractional, in microseconds)*
STMT_TEXT	CLOB(2 M)	stmt_text - SQL statement text
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
STATS_FABRICATE_TIME	BIGINT	The total time (in milliseconds) spent by system to create needed statistics without table or index scan during query compilation for a dynamic statement.
SYNC_RUNSTATS_TIME	BIGINT	The total time (in milliseconds) spent on synchronous statistics-collect activities during query compilation for a dynamic statement.

Table 181. Information returned by the SNAPDYN_SQL administrative view and the SNAP_GET_DYN_SQL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPFCM administrative view and SNAP_GET_FCM table function – Retrieve the fcm logical data group snapshot information

The SNAPFCM administrative view and the SNAP_GET_FCM table function return information about the fast communication manager from a database manager snapshot, in particular, the fcm logical data group.

SNAPFCM administrative view

Used with the SNAPDBM, SNAPDBM_MEMORY_POOL, SNAPFCM_PART and SNAPSWITCHES administrative views, the SNAPFCM administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 182 on page 643 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPFCM administrative view
- CONTROL privilege on the SNAPFCM administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_FCM table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information about the fast communication manager's message buffers on all database partitions.

```
SELECT BUFF_FREE, BUFF_FREE_BOTTOM, DBPARTITIONNUM
    FROM SYSIBMADM.SNAPFCM ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
BUFF_FREE        BUFF_FREE_BOTTOM     DBPARTITIONNUM
---------...---- -------------------- --------------
            5120                 5100              0
            5120                 5100              1
            5120                 5100              2
```

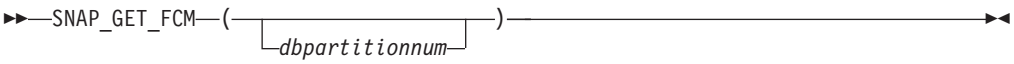
SNAP_GET_FCM table function

The SNAP_GET_FCM table function returns the same information as the SNAPFCM administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_V95, SNAP_GET_DBM_MEMORY_POOL, SNAP_GET_FCM_PART and SNAP_GET_SWITCHES table functions, the SNAP_GET_FCM table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 182 on page 643 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_FCM table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_FCM table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information about the fast communication manager's message buffers on database partition 1.

```
SELECT BUFF_FREE, BUFF_FREE_BOTTOM, DBPARTITIONNUM
   FROM TABLE(SYSPROC.SNAP_GET_FCM( 1 )) AS T
```

The following is an example of output from this query.

```
BUFF_FREE            BUFF_FREE_BOTTOM     DBPARTITIONNUM
-------------------- -------------------- --------------
                5120                 5100              1
```

Information returned

Table 182. Information returned by the SNAPFCM administrative view and the SNAP_GET_FCM table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BUFF_FREE	BIGINT	buff_free - FCM buffers currently free
BUFF_FREE_BOTTOM	BIGINT	buff_free_bottom - Minimum FCM Buffers Free
CH_FREE	BIGINT	ch_free - Channels Currently Free
CH_FREE_BOTTOM	BIGINT	ch_free_bottom - Minimum Channels Free
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPFCM_PART administrative view and SNAP_GET_FCM_PART table function – Retrieve the fcm_node logical data group snapshot information

The SNAPFCM_PART administrative view and the SNAP_GET_FCM_PART table function return information about the fast communication manager from a database manager snapshot, in particular, the fcm_node logical data group.

SNAPFCM_PART administrative view

Used with the SNAPDBM, SNAPDBM_MEMORY_POOL, SNAPFCM and SNAPSWITCHES administrative views, the SNAPFCM_PART administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 183 on page 646 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPFCM_PART administrative view
- CONTROL privilege on the SNAPFCM_PART administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_FCM_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve buffers sent and received information for the fast communication manager.

```
SELECT CONNECTION_STATUS, TOTAL_BUFFERS_SENT, TOTAL_BUFFERS_RECEIVED
   FROM SYSIBMADM.SNAPFCM_PART WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
CONNECTION_STATUS    TOTAL_BUFFERS_SENT   TOTAL_BUFFERS_RCVD
-------------------- -------------------- --------------------
INACTIVE                               2                    1

  1 record(s) selected.
```

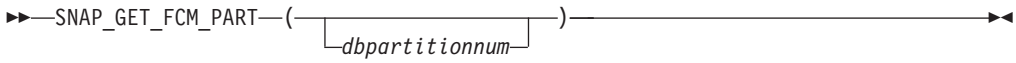
SNAP_GET_FCM_PART table function

The SNAP_GET_FCM_PART table function returns the same information as the SNAPFCM_PART administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_V95, SNAP_GET_DBM_MEMORY_POOL, SNAP_GET_FCM and SNAP_GET_SWITCHES table functions, the SNAP_GET_FCM_PART table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 183 on page 646 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_FCM_PART table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_FCM_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve buffers sent and received information for the fast communication manager for all database partitions.

```
SELECT FCM_DBPARTITIONNUM, TOTAL_BUFFERS_SENT, TOTAL_BUFFERS_RCVD,
   DBPARTITIONNUM FROM TABLE(SNAP_GET_FCM_PART()) AS T
   ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
FCM_DBPARTITIONNUM TOTAL_BUFFERS_SENT   TOTAL_BUFFERS_RCVD   DBPARTITIONNUM
------------------ -------------------- -------------------- --------------
                 0                  305                  305              0
                 1                 5647                 1664              0
                 2                 5661                 1688              0
                 0                   19                   19              1
                 1                  305                  301              1
                 2                 1688                 5661              1
```

```
0                1664              5647              2
1                  10                10              2
2                 301               305              2
```

Information returned

Table 183. Information returned by the SNAPFCM_PART administrative view and the SNAP_GET_FCM_PART table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
CONNECTION_STATUS	VARCHAR(10)	connection_status - Connection status. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • INACTIVE • ACTIVE • CONGESTED
TOTAL_BUFFERS_SENT	BIGINT	total_buffers_sent - Total FCM buffers sent
TOTAL_BUFFERS_RCVD	BIGINT	total_buffers_rcvd - Total FCM buffers received
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
FCM_DBPARTITIONNUM	SMALLINT	The database partition number to which data was sent or from which data was received (as per the TOTAL_BUFFERS_SENT and TOTAL_BUFFERS_RCVD columns).

SNAPHADR administrative view and SNAP_GET_HADR table function – Retrieve hadr logical data group snapshot information

The SNAPHADR administrative view and the SNAP_GET_HADR table function return information about high availability disaster recovery from a database snapshot, in particular, the hadr logical data group.

SNAPHADR administrative view

This administrative view allows you to retrieve hadr logical data group snapshot information for the currently connected database. The data is only returned by this view if the database is a primary or standby high availability disaster recovery (HADR) database.

Used with the SNAPDB, SNAPDB_MEMORY_POOL, SNAPDETAILLOG and SNAPSTORAGE_PATHS administrative views, the SNAPHADR administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 184 on page 649 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPHADR administrative view
- CONTROL privilege on the SNAPHADR administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_HADR table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the configuration and status information for HADR on the primary HADR database.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DBNAME, HADR_ROLE, HADR_STATE,
   HADR_SYNCMODE, HADR_CONNECT_STATUS
   FROM SYSIBMADM.SNAPHADR
```

The following is an example of output from this query.

```
DBNAME   HADR_ROLE HADR_STATE     HADR_SYNCMODE HADR_CONNECT_STATUS
-------- --------- -------------- ------------- -------------------
SAMPLE   PRIMARY   PEER           SYNC          CONNECTED

  1 record(s) selected.
```

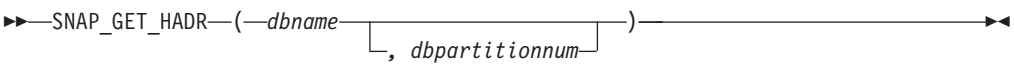
SNAP_GET_HADR table function

The SNAP_GET_HADR table function returns the same information as the SNAPHADR administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DB_V95, SNAP_GET_DB_MEMORY_POOL, SNAP_GET_DETAILLOG_V91 and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_HADR table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 184 on page 649 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to *NULL* and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_HADR table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_HADR table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the configuration and status information for HADR for all databases.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DBNAME, HADR_ROLE, HADR_STATE,
   HADR_SYNCMODE, HADR_CONNECT_STATUS
   FROM TABLE (SNAP_GET_HADR (CAST (NULL as VARCHAR(128)), 0)) as T
```

The following is an example of output from this query.

```
DBNAME   HADR_ROLE HADR_STATE     HADR_SYNCMODE HADR_CONNECT_STATUS
-------- --------- -------------- ------------- ------------------
SAMPLE   PRIMARY   PEER           SYNC          CONNECTED
TESTDB   PRIMARY   DISCONNECTED   NEARSYNC      DISCONNECTED

2 record(s) selected.
```

Information returned

Table 184. Information returned by the SNAPHADR administrative view and the SNAP_GET_HADR table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
HADR_ROLE	VARCHAR(10)	hadr_role - HADR role. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • PRIMARY • STANDARD • STANDBY
HADR_STATE	VARCHAR(14)	hadr_state - HADR state. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • DISCONNECTED • LOCAL_CATCHUP • PEER • REM_CATCH_PEN • REM_CATCHUP
HADR_SYNCMODE	VARCHAR(10)	hadr_syncmode - HADR synchronization mode. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • ASYNC • NEARSYNC • SYNC
HADR_CONNECT_STATUS	VARCHAR(12)	hadr_connect_status - HADR connection status. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • CONGESTED • CONNECTED • DISCONNECTED
HADR_CONNECT_TIME	TIMESTAMP	hadr_connect_time - HADR connection time
HADR_HEARTBEAT	INTEGER	hadr_heartbeat - HADR heartbeat
HADR_LOCAL_HOST	VARCHAR(255)	hadr_local_host - HADR local host

Table 184. Information returned by the SNAPHADR administrative view and the SNAP_GET_HADR table function (continued)

Column name	Data type	Description or corresponding monitor element
HADR_LOCAL_SERVICE	VARCHAR(40)	hadr_local_service - HADR local service
HADR_REMOTE_HOST	VARCHAR(255)	hadr_remote_host - HADR remote host
HADR_REMOTE_SERVICE	VARCHAR(40)	hadr_remote_service - HADR remote service
HADR_REMOTE_INSTANCE	VARCHAR(128)	hadr_remote_instance - HADR remote instance
HADR_TIMEOUT	BIGINT	hadr_timeout - HADR timeout
HADR_PRIMARY_LOG_FILE	VARCHAR(255)	hadr_primary_log_file - HADR primary log file
HADR_PRIMARY_LOG_PAGE	BIGINT	hadr_primary_log_page - HADR primary log page
HADR_PRIMARY_LOG_LSN	BIGINT	hadr_primary_log_lsn - HADR primary log LSN
HADR_STANDBY_LOG_FILE	VARCHAR(255)	hadr_standby_log_file - HADR standby log file
HADR_STANDBY_LOG_PAGE	BIGINT	hadr_standby_log_page - HADR standby log page
HADR_STANDBY_LOG_LSN	BIGINT	hadr_standby_log_lsn - HADR standby log LSN
HADR_LOG_GAP	BIGINT	hadr_log_gap - HADR log gap
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPLOCK administrative view and SNAP_GET_LOCK table function – Retrieve lock logical data group snapshot information

Note: This administrative view and table function have been deprecated and replaced by the "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415, "MON_GET_LOCKS - list all locks in the currently connected database" on page 442, and "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381.

The SNAPLOCK administrative view and the SNAP_GET_LOCK table function return snapshot information about locks, in particular, the lock logical data group.

SNAPLOCK administrative view

This administrative view allows you to retrieve lock logical data group snapshot information for the currently connected database.

Used with the SNAPLOCKWAIT administrative view, the SNAPLOCK administrative view provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 185 on page 653 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPLOCK administrative view
- CONTROL privilege on the SNAPLOCK administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_LOCK table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock information for the database partition 0 of the currently connected database.

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
   FROM SYSIBMADM.SNAPLOCK WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
AGENT_ID             LOCK_OBJECT_TYPE LOCK_MODE LOCK_STATUS
-------------------- ---------------- --------- -----------
                   7 TABLE            IX        GRNT

  1 record(s) selected.
```

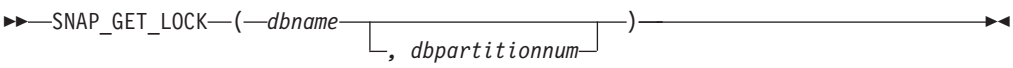
SNAP_GET_LOCK table function

The SNAP_GET_LOCK table function returns the same information as the SNAPLOCK administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_LOCKWAIT table function, the SNAP_GET_LOCK table function provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

Refer to Table 185 on page 653 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify a null value or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_LOCK table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_LOCK table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock information for the current database partition of the currently connected database.

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
   FROM TABLE(SNAP_GET_LOCK('',-1)) as T
```

The following is an example of output from this query.

```
AGENT_ID       LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS
--------...--- ------------------ ---------- -----------
           680 INTERNALV_LOCK     S          GRNT
           680 INTERNALP_LOCK     S          GRNT

2 record(s) selected.
```

Information returned

Table 185. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
TAB_FILE_ID	BIGINT	table_file_id - Table file identification
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK

Table 185. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_STATUS	VARCHAR(10)	lock_status - Lock status. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • CONV • GRNT
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
TABNAME	VARCHAR(128)	table_name - Table name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
LOCK_ATTRIBUTES	VARCHAR(128)	lock_attributes - Lock attributes. This interface returns a text identifier based on the defines in `sqlmon.h`. If there are no locks, the text identifier is NONE, otherwise, it is any combination of the following separated by a '+' sign: • ALLOW_NEW • DELETE_IN_BLOCK • ESCALATED • INSERT • NEW_REQUEST • RR • RR_IN_BLOCK • UPDATE_DELETE • WAIT_FOR_AVAIL
LOCK_COUNT	BIGINT	lock_count - Lock count

Table 185. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_CURRENT_MODE	VARCHAR(10)	lock_current_mode - Original lock mode before conversion. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_HOLD_COUNT	BIGINT	lock_hold_count - Lock hold count
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
LOCK_RELEASE_FLAGS	BIGINT	lock_release_flags - Lock release flags
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element is NULL.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function – Retrieve lockwait logical data group snapshot information

Note: This administrative view and table function have been deprecated and replaced by the "MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503 and the "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415, "MON_GET_LOCKS - list all locks in the currently connected database" on page 442, and "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381.

The SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function return snapshot information about lock waits, in particular, the lockwait logical data group.

SNAPLOCKWAIT administrative view

This administrative view allows you to retrieve lockwait logical data group snapshot information for the currently connected database.

Used with the SNAPLOCK administrative view, the SNAPLOCKWAIT administrative view provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 186 on page 658 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPLOCKWAIT administrative view
- CONTROL privilege on the SNAPLOCKWAIT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_LOCKWAIT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock wait information on database partition 0 for the currently connected database.

```
SELECT AGENT_ID, LOCK_MODE, LOCK_OBJECT_TYPE, AGENT_ID_HOLDING_LK,
   LOCK_MODE_REQUESTED FROM SYSIBMADM.SNAPLOCKWAIT
   WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
AGENT_ID     LOCK_MODE LOCK_OBJECT_TYPE ...
--------...- --------- ---------------- ...
           7 IX        TABLE            ...

  1 record(s) selected.
```

Output from this query (continued).

```
... AGENT_ID_HOLDING_LK  LOCK_MODE_REQUESTED
... -------------------- ------------------
...                   12 IS
```

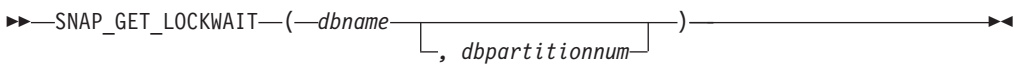
SNAP_GET_LOCKWAIT table function

The SNAP_GET_LOCKWAIT table function returns the same information as the SNAPLOCKWAIT administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_LOCK table function, the SNAP_GET_LOCKWAIT table function provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

Refer to Table 186 on page 658 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify a null value or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_LOCKWAIT table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_LOCKWAIT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock wait information on current database partition for the currently connected database.

```
SELECT AGENT_ID, LOCK_MODE, LOCK_OBJECT_TYPE, AGENT_ID_HOLDING_LK,
   LOCK_MODE_REQUESTED FROM TABLE(SNAP_GET_LOCKWAIT('',-1)) AS T
```

The following is an example of output from this query.

```
AGENT_ID      LOCK_MODE  LOCK_OBJECT_TYPE   ...
--------...-- ---------- ------------------ ...
          12 X          ROW_LOCK           ...

1 record(s) selected.
```

Output from this query (continued).

```
... AGENT_ID_HOLDING_LK  LOCK_MODE_REQUESTED
... -------------------- ------------------
...                    7 X
```

Usage note

To see lock wait information, you must first turn on the default LOCK monitor switch in the database manager configuration. To have the change take effect immediately explicitly attach to the instance using CLP and then issue the CLP command:

```
UPDATE DATABASE MANAGER CONFIGURATION CLP USING DFT_MON_LOCK ON
```

The default setting can also be turned on through the ADMIN_CMD stored procedure. For example:

```
CALL SYSPROC.ADMIN_CMD('update dbm cfg using DFT_MON_LOCK ON')
```

If the ADMIN_CMD stored procedure is used or if the clp command is used without having previously attached to the instance, the instance must be recycled before the change takes effect.

Information returned

Table 186. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
SUBSECTION_NUMBER	BIGINT	ss_number - Subsection number

Table 186. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
AGENT_ID_HOLDING_LK	BIGINT	agent_id_holding_lock - Agent ID holding lock

Table 186. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_WAIT_START_TIME	TIMESTAMP	lock_wait_start_time - Lock wait start timestamp
LOCK_MODE_REQUESTED	VARCHAR(10)	lock_mode_requested - Lock mode requested. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
TABNAME	VARCHAR(128)	table_name - Table name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
APPL_ID_HOLDING_LK	VARCHAR(128)	appl_id_holding_lk - Application ID holding lock
LOCK_ATTRIBUTES	VARCHAR(128)	lock_attributes - Lock attributes. This interface returns a text identifier based on the defines in `sqlmon.h`. If there are no locks, the text identifier is NONE, otherwise, it is any combination of the following separated by a '+' sign: • ALLOW_NEW • DELETE_IN_BLOCK • ESCALATED • INSERT • NEW_REQUEST • RR • RR_IN_BLOCK • UPDATE_DELETE • WAIT_FOR_AVAIL

Table 186. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_CURRENT_MODE	VARCHAR(10)	lock_current_mode - Original lock mode before conversion. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
LOCK_RELEASE_FLAGS	BIGINT	lock_release_flags - Lock release flags.
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element is NULL.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPSTMT administrative view and SNAP_GET_STMT table function – Retrieve statement snapshot information

The SNAPSTMT administrative view and the SNAP_GET_STMT table function return information about SQL or XQuery statements from an application snapshot.

SNAPSTMT administrative view

This administrative view allows you to retrieve statement snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPAPPL_INFO and SNAPSUBSECTION administrative views, the SNAPSTMT administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS on database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 187 on page 664 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:
- SELECT privilege on the SNAPSTMT administrative view
- CONTROL privilege on the SNAPSTMT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the SNAP_GET_STMT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve rows read, written and operation performed for statements executed on the currently connected single-partition database.

```
SELECT SUBSTR(STMT_TEXT,1,30) AS STMT_TEXT, ROWS_READ, ROWS_WRITTEN,
   STMT_OPERATION FROM SYSIBMADM.SNAPSTMT
```

The following is an example of output from this query.

```
STMT_TEXT     ROWS_READ     ROWS_WRITTEN     STMT_OPERATION
---------...- ---------...- ------------...- --------------------
-                         0                0 FETCH
-                         0                0 STATIC_COMMIT

  2 record(s) selected.
```

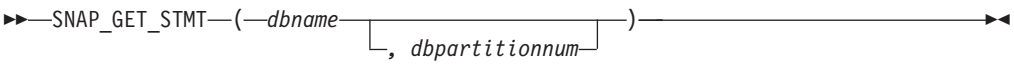
SNAP_GET_STMT table function

The SNAP_GET_STMT table function returns the same information as the SNAPSTMT administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95 and SNAP_GET_SUBSECTION table functions, the SNAP_GET_STMT table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 187 on page 664 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_STMT table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_STMT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve rows read, written and operation performed for statements executed on current database partition of currently connected database.

```
SELECT SUBSTR(STMT_TEXT,1,30) AS STMT_TEXT, ROWS_READ,
   ROWS_WRITTEN, STMT_OPERATION FROM TABLE(SNAP_GET_STMT('',-1)) AS T
```

The following is an example of output from this query.

```
STMT_TEXT                      ROWS_READ     ...
------------------------------ ---------...- ...
update t set a=3                          0 ...
SELECT SUBSTR(STMT_TEXT,1,30)             0 ...
-                                         0 ...
-                                         0 ...
```

```
update t set a=2                          9 ...
                                            ...
5 record(s) selected.                       ...
```

Output from this query (continued).

```
... ROWS_WRITTEN     STMT_OPERATION
... ------------...- --------------------
...                0 EXECUTE_IMMEDIATE
...                0 FETCH
...                0 NONE
...                0 NONE
...                1 EXECUTE_IMMEDIATE
...
```

Information returned

Table 187. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
NUM_AGENTS	BIGINT	num_agents - Number of agents working on a statement
AGENTS_TOP	BIGINT	agents_top - Number of agents created
STMT_TYPE	VARCHAR(20)	stmt_type - Statement type. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • DYNAMIC • NON_STMT • STATIC • STMT_TYPE_UNKNOWN

Table 187. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function (continued)

Column name	Data type	Description or corresponding monitor element
STMT_OPERATION	VARCHAR(20)	stmt_operation/operation - Statement operation. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • CALL • CLOSE • COMPILE • DESCRIBE • EXECUTE • EXECUTE_IMMEDIATE • FETCH • FREE_LOCATOR • GETAA • GETNEXTCHUNK • GETTA • NONE • OPEN • PREP_COMMIT • PREP_EXEC • PREP_OPEN • PREPARE • REBIND • REDIST • REORG • RUNSTATS • SELECT • SET • STATIC_COMMIT • STATIC_ROLLBACK
SECTION_NUMBER	BIGINT	section_number - Section number
QUERY_COST_ESTIMATE	BIGINT	query_cost_estimate - Query cost estimate
QUERY_CARD_ESTIMATE	BIGINT	query_card_estimate - Query number of rows estimate
DEGREE_PARALLELISM	BIGINT	degree_parallelism - Degree of parallelism
STMT_SORTS	BIGINT	stmt_sorts - Statement sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated

Table 187. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function (continued)

Column name	Data type	Description or corresponding monitor element
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
FETCH_COUNT	BIGINT	fetch_count - Number of successful fetches
STMT_START	TIMESTAMP	stmt_start - Statement operation start timestamp
STMT_STOP	TIMESTAMP	stmt_stop - Statement operation stop timestamp
STMT_USR_CPU_TIME_S	BIGINT	stmt_usr_cpu_time - User CPU time used by statement (in seconds)*
STMT_USR_CPU_TIME_MS	BIGINT	stmt_usr_cpu_time - User CPU time used by statement (fractional, in microseconds)*
STMT_SYS_CPU_TIME_S	BIGINT	stmt_sys_cpu_time - System CPU time used by statement (in seconds)*
STMT_SYS_CPU_TIME_MS	BIGINT	stmt_sys_cpu_time - System CPU time used by statement (fractional, in microseconds)*
STMT_ELAPSED_TIME_S	BIGINT	stmt_elapsed_time - Most recent statement elapsed time (in seconds)*
STMT_ELAPSED_TIME_MS	BIGINT	stmt_elapsed_time - Most recent statement elapsed time (fractional, in microseconds)*
BLOCKING_CURSOR	SMALLINT	blocking_cursor - Blocking cursor
STMT_NODE_NUMBER	SMALLINT	stmt_node_number - Statement node
CURSOR_NAME	VARCHAR(128)	cursor_name - Cursor name
CREATOR	VARCHAR(128)	creator - Application creator
PACKAGE_NAME	VARCHAR(128)	package_name - Package name
STMT_TEXT	CLOB(16 M)	stmt_text - SQL statement text
CONSISTENCY_TOKEN	VARCHAR(128)	consistency_token - Package consistency token
PACKAGE_VERSION_ID	VARCHAR(128)	package_version_id - Package version
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads

Table 187. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads monitor element
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads monitor element
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPSTORAGE_PATHS administrative view and SNAP_GET_STORAGE_PATHS_V97 table function - Retrieve automatic storage path information

The SNAPSTORAGE_PATHS administrative view and the SNAP_GET_STORAGE_PATHS_V97 table function return a list of automatic storage paths for the database including file system information for each storage path, specifically, from the db_storage_group logical data group.

SNAPSTORAGE_PATHS administrative view

This administrative view allows you to retrieve automatic storage path information for the currently connected database.

Used with the SNAPDB, SNAPDETAILLOG, SNAPHADR and SNAPDB_MEMORY_POOL administrative views, the SNAPSTORAGE_PATHS

administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 188 on page 669 for a complete list of information that can be returned.

Authorization

- SYSMON authority
- SELECT or CONTROL privilege on the SNAPSTORAGE_PATHS administrative view and EXECUTE privilege on the SNAP_GET_STORAGE_PATHS_V97 table function.

Example

Retrieve the storage path for the currently connected single-partition database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(DB_STORAGE_PATH,1,8)
   AS DB_STORAGE_PATH, SUBSTR(HOSTNAME,1,10) AS HOSTNAME
   FROM SYSIBMADM.SNAPSTORAGE_PATHS
```

The following is an example of output from this query.

```
DB_NAME  DB_STORAGE_PATH HOSTNAME
-------- --------------- ----------
STOPATH  d:              JESSICAE

  1 record(s) selected.
```

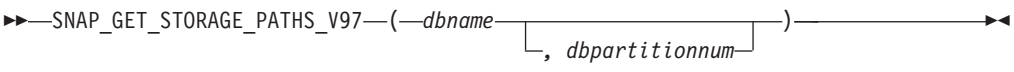
SNAP_GET_STORAGE_PATHS_V97 table function

The SNAP_GET_STORAGE_PATHS_V97 table function returns the same information as the SNAPSTORAGE_PATHS administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DB_V95, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_DB_MEMORY_POOL table functions, the SNAP_GET_STORAGE_PATHS_V97 table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 188 on page 669 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as

returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL,-1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_STORAGE_PATHS_V97 table function takes a snapshot for the currently connected database and database partition number.

Authorization

- SYSMON authority
- EXECUTE privilege on the SNAP_GET_STORAGE_PATHS_V97 table function.

Examples

Retrieve the storage path information for all active databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, DB_STORAGE_PATH
   FROM TABLE(SNAP_GET_STORAGE_PATHS_V97(CAST (NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  DB_STORAGE_PATH
-------- ------------------...
STOPATH  /home/jessicae/sdb
MYDB     /home/jessicae/mdb

  2 record(s) selected
```

Information returned

The BUFFERPOOL monitor switch must be turned on in order for the file system information to be returned.

Table 188. Information returned by the SNAPSTORAGE_PATHS administrative view and the SNAP_GET_STORAGE_PATHS_V97 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
DB_STORAGE_PATH	VARCHAR(256)	db_storage_path - Automatic storage path

Table 188. Information returned by the SNAPSTORAGE_PATHS administrative view and the SNAP_GET_STORAGE_PATHS_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
DB_STORAGE_PATH_WITH_DPE	VARCHAR(256)	Automatic storage path containing the unevaluated database partition expression (DPE). Returns NULL if the storage path does not contain a DPE.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
DB_STORAGE_PATH_STATE	VARCHAR(16)	Automatic storage path state (values can currently be one of "IN_USE", "NOT_IN_USE", "DROP_PENDING").
FS_ID	VARCHAR(22)	fs_id - Unique file system identification number
FS_TOTAL_SIZE	BIGINT	fs_total_size - Total size of a file system
FS_USED_SIZE	BIGINT	fs_used_size - Amount of space used on a file system
STO_PATH_FREE_SIZE	BIGINT	sto_path_free_sz - Automatic storage path free space

SNAPSUBSECTION administrative view and SNAP_GET_SUBSECTION table function – Retrieve subsection logical monitor group snapshot information

The SNAPSUBSECTION administrative view and the SNAP_GET_SUBSECTION table function return information about application subsections, namely the subsection logical monitor grouping.

SNAPSUBSECTION administrative view

This administrative view allows you to retrieve subsection logical monitor group snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPAPPL_INFO and SNAPSTMT administrative views, the SNAPSUBSECTION administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS on database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 189 on page 673 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPSUBSECTION administrative view

- CONTROL privilege on the SNAPSUBSECTION administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the SNAP_GET_SUBSECTION table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Get status for subsections executing on all database partitions.

```
SELECT DB_NAME, STMT_TEXT, SS_STATUS, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPSUBSECTION
   ORDER BY DB_NAME, SS_STATUS, DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME     STMT_TEXT                 SS_STATUS     DBPARTITIONNUM
--------...- ----------------------...- ---------...- --------------
SAMPLE      select * from EMPLOYEE     EXEC                       0
SAMPLE      select * from EMPLOYEE     EXEC                       1
```

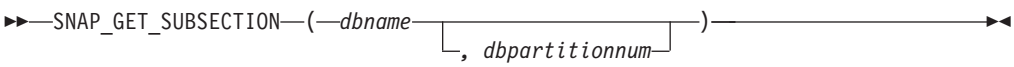
SNAP_GET_SUBSECTION table function

The SNAP_GET_SUBSECTION table function returns the same information as the SNAPSUBSECTION administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Refer to Table 189 on page 673 for a complete list of information that can be returned.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95 and SNAP_GET_STMT table functions, the SNAP_GET_SUBSECTION table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database

name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_SUBSECTION table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_SUBSECTION table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Get status for subsections executing on all database partitions.

```
SELECT DB_NAME, STMT_TEXT, SS_STATUS, DBPARTITIONNUM
   FROM TABLE(SYSPROC.SNAP_GET_SUBSECTION( '', 0 )) as T
   ORDER BY DB_NAME, SS_STATUS, DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME     STMT_TEXT                 SS_STATUS     DBPARTITIONNUM
--------...- ----------------------...- ---------...- --------------
SAMPLE      select * from EMPLOYEE     EXEC                       0
SAMPLE      select * from EMPLOYEE     EXEC                       1
```

Information returned

Table 189. Information returned by the SNAPSUBSECTION administrative view and the SNAP_GET_SUBSECTION table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
STMT_TEXT	CLOB(16 M)	stmt_text - SQL statement text
SS_EXEC_TIME	BIGINT	ss_exec_time - Subsection execution elapsed time
TQ_TOT_SEND_SPILLS	BIGINT	tq_tot_send_spills - Total number of table queue buffers overflowed
TQ_CUR_SEND_SPILLS	BIGINT	tq_cur_send_spills - Current number of table queue buffers overflowed
TQ_MAX_SEND_SPILLS	BIGINT	tq_max_send_spills - Maximum number of table queue buffers overflows
TQ_ROWS_READ	BIGINT	tq_rows_read - Number of rows read from table queues
TQ_ROWS_WRITTEN	BIGINT	tq_rows_written - Number of rows written to table queues
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
SS_USR_CPU_TIME_S	BIGINT	ss_usr_cpu_time - User CPU time used by subsection (in seconds)*
SS_USR_CPU_TIME_MS	BIGINT	ss_usr_cpu_time - User CPU time used by subsection (fractional, in microseconds)*
SS_SYS_CPU_TIME_S	BIGINT	ss_sys_cpu_time - System CPU time used by subsection (in seconds)*
SS_SYS_CPU_TIME_MS	BIGINT	ss_sys_cpu_time - System CPU time used by subsection (fractional, in microseconds)*
SS_NUMBER	INTEGER	ss_number - Subsection number
SS_STATUS	VARCHAR(20)	ss_status - Subsection status. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • EXEC • TQ_WAIT_TO_RCV • TQ_WAIT_TO_SEND • COMPLETED
SS_NODE_NUMBER	SMALLINT	ss_node_number - Subsection node number
TQ_NODE_WAITED_FOR	SMALLINT	tq_node_waited_for - Waited for node on a table queue

Table 189. Information returned by the SNAPSUBSECTION administrative view and the SNAP_GET_SUBSECTION table function (continued)

Column name	Data type	Description or corresponding monitor element
TQ_WAIT_FOR_ANY	INTEGER	tq_wait_for_any - Waiting for any node to send on a table queue
TQ_ID_WAITING_ON	INTEGER	tq_id_waiting_on - Waited on node on a table queue
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPSWITCHES administrative view and SNAP_GET_SWITCHES table function – Retrieve database snapshot switch state information

The SNAPSWITCHES administrative view and the SNAP_GET_SWITCHES table function return information about the database snapshot switch state.

SNAPSWITCHES administrative view

This view provides the data equivalent to the GET DBM MONITOR SWITCHES CLP command.

The schema is SYSIBMADM.

Refer to Table 190 on page 676 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPSWITCHES administrative view
- CONTROL privilege on the SNAPSWITCHES administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_SWITCHES table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve DBM monitor switches state information for all database partitions.

```
SELECT UOW_SW_STATE, STATEMENT_SW_STATE, TABLE_SW_STATE, BUFFPOOL_SW_STATE,
   LOCK_SW_STATE, SORT_SW_STATE, TIMESTAMP_SW_STATE,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPSWITCHES
```

The following is an example of output from this query.

```
UOW_SW_STATE STATEMENT_SW_STATE TABLE_SW_STATE BUFFPOOL_SW_STATE ...
------------ ------------------ -------------- ----------------- ...
           0                  0              0                 0 ...
           0                  0              0                 0 ...
           0                  0              0                 0 ...
                                                                  ...
  3 record selected.
```

Output from this query (continued).

```
... LOCK_SW_STATE SORT_SW_STATE TIMESTAMP_SW_STATE DBPARTITIONNUM
... ------------- ------------- ------------------ --------------
...             1             0                  1              0
...             1             0                  1              1
...             1             0                  1              2
```

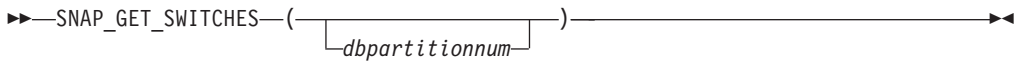
SNAP_GET_SWITCHES table function

The SNAP_GET_SWITCHES table function returns the same information as the SNAPSWITCHES administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

This table function provides the data equivalent to the GET DBM MONITOR SWITCHES CLP command.

Refer to Table 190 on page 676 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the

SNAP_GET_SWITCHES table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_SWITCHES table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve DBM monitor switches state information for the current database partition.

```
SELECT UOW_SW_STATE, STATEMENT_SW_STATE, TABLE_SW_STATE,
   BUFFPOOL_SW_STATE,LOCK_SW_STATE, SORT_SW_STATE, TIMESTAMP_SW_STATE
   FROM TABLE(SNAP_GET_SWITCHES(-1)) AS T
```

The following is an example of output from this query.

```
UOW_SW_STATE STATEMENT_SW_STATE TABLE_SW_STATE...
------------ ----------------- --------------...
           1                 1              1...
                                             ...
1 record(s) selected.                        ...
```

Output from this query (continued).

```
... BUFFPOOL_SW_STATE LOCK_SW_STATE SORT_SW_STATE TIMESTAMP_SW_STATE
... ---------------- ------------- ------------- ------------------
...                1             1             0                  1
```

Information returned

Table 190. Information returned by the SNAPSWITCHES administrative view and the SNAP_GET_SWITCHES table function

Column name	Data type	Description
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
UOW_SW_STATE	SMALLINT	State of the unit of work monitor recording switch (0 or 1).
UOW_SW_TIME	TIMESTAMP	If the unit of work monitor recording switch is on, the date and time that this switch was turned on.
STATEMENT_SW_STATE	SMALLINT	State of the SQL statement monitor recording switch (0 or 1).

Table 190. Information returned by the SNAPSWITCHES administrative view and the SNAP_GET_SWITCHES table function (continued)

Column name	Data type	Description
STATEMENT_SW_TIME	TIMESTAMP	If the SQL statement monitor recording switch is on, the date and time that this switch was turned on.
TABLE_SW_STATE	SMALLINT	State of the table activity monitor recording switch (0 or 1).
TABLE_SW_TIME	TIMESTAMP	If the table activity monitor recording switch is on, the date and time that this switch was turned on.
BUFFPOOL_SW_STATE	SMALLINT	State of the buffer pool activity monitor recording switch (0 or 1).
BUFFPOOL_SW_TIME	TIMESTAMP	If the buffer pool activity monitor recording switch is on, the date and time that this switch was turned on.
LOCK_SW_STATE	SMALLINT	State of the lock monitor recording switch (0 or 1).
LOCK_SW_TIME	TIMESTAMP	If the lock monitor recording switch is on, the date and time that this switch was turned on.
SORT_SW_STATE	SMALLINT	State of the sorting monitor recording switch (0 or 1).
SORT_SW_TIME	TIMESTAMP	If the sorting monitor recording switch is on, the date and time that this switch was turned on.
TIMESTAMP_SW_STATE	SMALLINT	State of the timestamp monitor recording switch (0 or 1)
TIMESTAMP_SW_TIME	TIMESTAMP	If the timestamp monitor recording switch is on, the date and time that this switch was turned on.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTAB administrative view and SNAP_GET_TAB_V91 table function - Retrieve table logical data group snapshot information

The SNAPTAB administrative view and the SNAP_GET_TAB_V91 table function return snapshot information from the table logical data group.

SNAPTAB administrative view

This administrative view allows you to retrieve table logical data group snapshot information for the currently connected database.

Used in conjunction with the SNAPTAB_REORG administrative view, the SNAPTAB administrative view returns equivalent information to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 191 on page 680 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTAB administrative view
- CONTROL privilege on the SNAPTAB administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TAB_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the schema and name for all active tables.

```
SELECT SUBSTR(TABSCHEMA,1,8), SUBSTR(TABNAME,1,15) AS TABNAME, TAB_TYPE,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPTAB
```

The following is an example of output from this query.

```
TABSCHEMA TABNAME         TAB_TYPE     DBPARTITIONNUM
--------- --------------- ------------ --------------
SYSTOOLS  HMON_ATM_INFO   USER_TABLE                0

  1 record selected.
```

SNAP_GET_TAB_V91 table function

The SNAP_GET_TAB_V91 table function returns the same information as the SNAPTAB administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_TAB_REORG table function, the SNAP_GET_TAB_V91 table function returns equivalent information to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

Refer to Table 191 on page 680 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify `NULL` or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TAB_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TAB_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of active tables as an aggregate view for the currently connected database.

```
SELECT SUBSTR(TABSCHEMA,1,8) AS TABSCHEMA, SUBSTR(TABNAME,1,15) AS TABNAME,
   TAB_TYPE, DBPARTITIONNUM FROM TABLE(SNAP_GET_TAB('',-2)) AS T
```

The following is an example of output from this query.

```
TABSCHEMA TABNAME         TAB_TYPE      DBPARTITIONNUM
--------- --------------- ------------- --------------
SYSTOOLS  HMON_ATM_INFO   USER_TABLE                 -
JESSICAE  EMPLOYEE        USER_TABLE                 -
```

Information returned

Table 191. Information returned by the SNAPTAB administrative view and the SNAP_GET_TAB_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
TAB_FILE_ID	BIGINT	table_file_id - Table file identification
TAB_TYPE	VARCHAR(14)	table_type - Table type. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • USER_TABLE • DROPPED_TABLE • TEMP_TABLE • CATALOG_TABLE • REORG_TABLE
DATA_OBJECT_PAGES	BIGINT	data_object_pages - Data object pages
INDEX_OBJECT_PAGES	BIGINT	index_object_pages - Index object pages
LOB_OBJECT_PAGES	BIGINT	lob_object_pages - LOB object pages
LONG_OBJECT_PAGES	BIGINT	long_object_pages - Long object pages
XDA_OBJECT_PAGES	BIGINT	xda_object_pages - XDA Object Pages
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
OVERFLOW_ACCESSES	BIGINT	overflow_accesses - Accesses to overflowed records
PAGE_REORGS	BIGINT	page_reorgs - Page reorganizations
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
TBSP_ID	BIGINT	tablespace_id - Table space identification
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element will be NULL.

SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function - Retrieve table reorganization snapshot information

The SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function return table reorganization information. If no tables have been reorganized, 0 rows are returned. When a data partitioned table is reorganized, one record for each data partition is returned. If only a specific data partition of a data partitioned table is reorganized, only a record the for the partition is returned.

SNAPTAB_REORG administrative view

This administrative view allows you to retrieve table reorganization snapshot information for the currently connected database.

Used with the SNAPTAB administrative view, the SNAPTAB_REORG administrative view provides the data equivalent to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 192 on page 684 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTAB_REORG administrative view
- CONTROL privilege on the SNAPTAB_REORG administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TAB_REORG table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Select details on reorganization operations for all database partitions on the currently connected database.

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
   AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
   REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TAB_NAME     TAB_SCHEMA     REORG_PHASE      ...
--------...- ----------...- ---------------- ...
EMPLOYEE     DBUSER         REPLACE          ...
EMPLOYEE     DBUSER         REPLACE          ...
EMPLOYEE     DBUSER         REPLACE          ...
                                             ...
3 record(s) selected.
```

Output from this query (continued).

```
... REORG_TYPE           REORG_STATUS REORG_COMPLETION DBPARTITIONNUM
... -------------------- ------------ ---------------- --------------
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       0
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       1
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       2
```

Select all information about a reorganization operation to reclaim extents from a multidimensional clustering (MDC) table from the SNAPTAB_REORG administrative view..

```
db2 -v "select * from sysibmadm.snaptab_reorg"

TABNAME  REORG_PHASE       REORG_MAX_PHASE   REORG_TYPE
-------- ----------------- ----------------- -------------------------
T1       RELEASE           3                 RECLAIM_EXTENTS+ALLOW_WRITE

REORG_STATUS REORG_COMPLETION REORG_START                REORG_END
------------ ---------------- --------------------------- --------------------------
COMPLETED    SUCCESS          2008-09-24-14.35.30.734741 2008-09-24-14.35.31.460674
```

SNAP_GET_TAB_REORG table function

The SNAP_GET_TAB_REORG table function returns the same information as the SNAPTAB_REORG administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TAB table function, the SNAP_GET_TAB_REORG table function provides the data equivalent to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

Refer to Table 192 on page 684 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TAB_REORG table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TAB_REORG table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Select details on reorganization operations for database partition 1 on the currently connected database.

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
   AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
   REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
   FROM TABLE( SNAP_GET_TAB_REORG('', 1)) AS T
```

The following is an example of output from this query.

```
TAB_NAME     TAB_SCHEMA     REORG_PHASE     REORG_TYPE           ...
--------...- ----------...- ----------...- -------------------- ...
EMPLOYEE     DBUSER         REPLACE         RECLAIM+OFFLINE+ALLO ...
                                                                 ...
  1 record(s) selected.                                          ...
```

Output from this query (continued).

```
... REORG_STATUS REORG_COMPLETION DBPARTITIONNUM
... ------------ ---------------- --------------
... COMPLETED    SUCCESS                     1
...
```

Select all information about a reorganization operation to reclaim extents from a multidimensional clustering (MDC) table using the SNAP_GET_TAB_REORG table function.

```
db2 -v "select * from table(snap_get_tab_reorg(''))"

TABNAME  REORG_PHASE       REORG_MAX_PHASE   REORG_TYPE
-------- ----------------- ----------------- --------------------------
T1       RELEASE           3                 RECLAIM_EXTENTS+ALLOW_WRITE

REORG_STATUS REORG_COMPLETION REORG_START                REORG_END
------------ ---------------- -------------------------- --------------------------
COMPLETED    SUCCESS          2008-09-24-14.35.30.734741 2008-09-24-14.35.31.460674
```

Information returned

Table 192. Information returned by the SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABNAME	VARCHAR (128)	table_name - Table name
TABSCHEMA	VARCHAR (128)	table_schema - Table schema name
PAGE_REORGS	BIGINT	page_reorgs - Page reorganizations
REORG_PHASE	VARCHAR (16)	reorg_phase - Table reorganize phase. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • BUILD • DICT_SAMPLE • INDEX_RECREATE • REPLACE • SORT • SCAN • DRAIN • RELEASE or SORT+DICT_SAMPLE.
REORG_MAX_PHASE	INTEGER	reorg_max_phase - Maximum table reorganize phase
REORG_CURRENT_ COUNTER	BIGINT	reorg_current_counter - Table reorganize progress
REORG_MAX_COUNTER	BIGINT	reorg_max_counter - Total amount of table reorganization

Table 192. Information returned by the SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function (continued)

Column name	Data type	Description or corresponding monitor element

REORG_TYPE	VARCHAR (128)	reorg_type - Table reorganize attributes. This interface returns a text identifier using a combination of the following identifiers separated by '+':
		
		Either:
		• RECLAIM
		• RECLUSTER
		• RECLAIM_EXTS
		
		and either:
		• +OFFLINE
		• +ONLINE
		
		If access mode is specified, it is one of:
		• +ALLOW_NONE
		• +ALLOW_READ
		• +ALLOW_WRITE
		
		If offline and RECLUSTER option, one of:
		• +INDEXSCAN
		• +TABLESCAN
		
		If offline, one of:
		• +LONGLOB
		• +DATAONLY
		
		If offline, and option is specified, any of:
		• +CHOOSE_TEMP
		• +KEEPDICTIONARY
		• +RESETDICTIONARY
		
		If online, and option is specified:
		• +NOTRUNCATE
		
		Example 1: If a REORG TABLE TEST.EMPLOYEE was run, the following would be displayed:
		`RECLAIM+OFFLINE+ALLOW_READ+DATAONLY +KEEPDICTIONARY`
		
		Example 2: If a REORG TABLE TEST.EMPLOYEE INDEX EMPIDX INDEXSCAN was run, then the following would be displayed:
		`RECLUSTER+OFFLINE+ALLOW_READ+INDEXSCAN +DATAONLY+KEEPDICTIONARY`

Table 192. Information returned by the SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function (continued)

Column name	Data type	Description or corresponding monitor element
REORG_STATUS	VARCHAR (10)	reorg_status - Table reorganize status. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • COMPLETED • PAUSED • STARTED • STOPPED • TRUNCATE
REORG_COMPLETION	VARCHAR (10)	reorg_completion - Table reorganization completion flag. This interface returns a text identifier, based on defines in `sqlmon.h` and is one of: • FAIL • SUCCESS
REORG_START	TIMESTAMP	reorg_start - Table reorganize start time
REORG_END	TIMESTAMP	reorg_end - Table reorganize end time
REORG_PHASE_START	TIMESTAMP	reorg_phase_start - Table reorganize phase start time
REORG_INDEX_ID	BIGINT	reorg_index_id - Index used to reorganize the table
REORG_TBSPC_ID	BIGINT	reorg_tbspc_id - Table space where table is reorganized
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element will be NULL.
REORG_ ROWSCOMPRESSED	BIGINT	reorg_rows_compressed - Rows compressed
REORG_ROWSREJECTED	BIGINT	reorg_rows_rejected_for_compression - Rows rejected for compression
REORG_LONG_TBSPC_ID	BIGINT	reorg_long_tbspc_id - Table space where long objects are reorganized

SNAPTBSP administrative view and SNAP_GET_TBSP_V91 table function - Retrieve table space logical data group snapshot information

The SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function return snapshot information from the table space logical data group.

SNAPTBSP administrative view

This administrative view allows you to retrieve table space logical data group snapshot information for the currently connected database.

Used in conjunction with the SNAPTBSP_PART, SNAPTBSP_QUIESCER, SNAPTBSP_RANGE, SNAPCONTAINER administrative views, the SNAPTBSP administrative view returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 193 on page 689 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTBSP administrative view
- CONTROL privilege on the SNAPTBSP administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TBSP_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of table spaces on the catalog database partition for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME,1,30) AS TBSP_NAME, TBSP_ID, TBSP_TYPE,
   TBSP_CONTENT_TYPE FROM SYSIBMADM.SNAPTBSP WHERE DBPARTITIONNUM = 1
```

The following is an example of output from this query.

```
TBSP_NAME  TBSP_ID     TBSP_TYPE  TBSP_CONTENT_TYPE
---------- -------...- ---------- -----------------
TEMPSPACE1           1 SMS        SYSTEMP
USERSPACE1           2 DMS        LONG

2 record(s) selected.
```

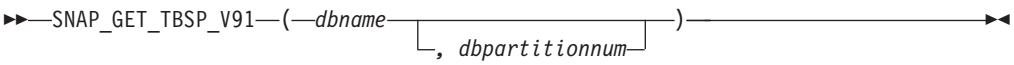
SNAP_GET_TBSP_V91 table function

The SNAP_GET_TBSP_V91 table function returns the same information as the SNAPTBSP administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_QUIESCER, SNAP_GET_TBSP_RANGE, SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_V91 table function returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 193 on page 689 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify `NULL` or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TBSP_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of table spaces for all database partitions for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME,1,10) AS TBSP_NAME, TBSP_ID, TBSP_TYPE,
   TBSP_CONTENT_TYPE, DBPARTITIONNUM FROM TABLE(SNAP_GET_TBSP_V91('')) AS T
```

The following is an example of output from this query.

```
TBSP_NAME  TBSP_ID     TBSP_TYPE  TBSP_CONTENT_TYPE DBPARTITIONNUM
---------- -------...- ---------- ----------------- -------------
TEMPSPACE1           1 SMS        SYSTEMP                         1
USERSPACE1           2 DMS        LONG                            1
SYSCATSPAC           0 DMS        ANY                             0
TEMPSPACE1           1 SMS        SYSTEMP                         0
USERSPACE1           2 DMS        LONG                            0
SYSTOOLSPA           3 DMS        LONG                            0
TEMPSPACE1           1 SMS        SYSTEMP                         2
USERSPACE1           2 DMS        LONG                            2

  8 record(s) selected.
```

Information returned

Table 193. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
TBSP_TYPE	VARCHAR(10)	tablespace_type - Table space type. This interface returns a text identifier based on defines in `sqlutil.h`, and is one of: • DMS • SMS
TBSP_CONTENT_TYPE	VARCHAR(10)	tablespace_content_type - Table space contents type. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ANY • LARGE • SYSTEMP • USRTEMP
TBSP_PAGE_SIZE	BIGINT	tablespace_page_size - Table space page size
TBSP_EXTENT_SIZE	BIGINT	tablespace_extent_size - Table space extent size
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size

Table 193. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_CUR_POOL_ID	BIGINT	tablespace_cur_pool_id - Buffer pool currently being used
TBSP_NEXT_POOL_ID	BIGINT	tablespace_next_pool_id - Buffer pool that will be used at next startup
FS_CACHING	SMALLINT	fs_caching - File system caching
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads

Table 193. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
FILES_CLOSED	BIGINT	files_closed - Database files closed
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages

Table 193. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_REBALANCER_MODE	VARCHAR(10)	tablespace_rebalancer_mode - Rebalancer mode. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • NO_REBAL • FWD_REBAL • REV_REBAL
TBSP_USING_AUTO_STORAGE	SMALLINT	tablespace_using_auto_storage - Table space enabled for automatic storage
TBSP_AUTO_RESIZE_ENABLED	SMALLINT	tablespace_auto_resize_enabled - Table space automatic resizing enabled
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V97 table function - Retrieve tablespace_nodeinfo logical data group snapshot information

The SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V97 table function return snapshot information from the tablespace_nodeinfo logical data group.

SNAPTBSP_PART administrative view

This administrative view allows you to retrieve tablespace_nodeinfo logical data group snapshot information for the currently connected database.

Used in conjunction with the SNAPTBSP, SNAPTBSP_QUIESCER, SNAPTBSP_RANGE, SNAPCONTAINER administrative views, the SNAPTBSP_PART administrative view returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 194 on page 694 for a complete list of information that can be returned.

Authorization

- SYSMON authority
- SELECT or CONTROL privilege on the SNAPTBSP_PART administrative view and EXECUTE privilege on the SNAP_GET_TBSP_PART_V97 table function.

Example

Retrieve a list of table spaces and their state for all database partitions of the currently connected database.

```
SELECT SUBSTR(TBSP_NAME,1,30) AS TBSP_NAME, TBSP_ID,
   SUBSTR(TBSP_STATE,1,30) AS TBSP_STATE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTBSP_PART
```

The following is an example of output from this query.

```
TBSP_NAME       TBSP_ID     TBSP_STATE     DBPARTITIONNUM
----------...- -------...- ----------...- --------------
SYSCATSPACE               0 NORMAL                     0
TEMPSPACE1                1 NORMAL                     0
USERSPACE1                2 NORMAL                     0
TEMPSPACE1                1 NORMAL                     1
USERSPACE1                2 NORMAL                     1

   5 record(s) selected.
```

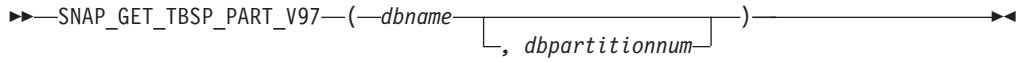
SNAP_GET_TBSP_PART_V97 table function

The SNAP_GET_TBSP_PART_V97 table function returns the same information as the SNAPTBSP_PART administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_TBSP_V97, SNAP_GET_TBSP_QUIESCER, SNAP_GET_TBSP_RANGE, SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_PART_V97 table function returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 194 on page 694 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could

have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_PART_V97 table function takes a snapshot for the currently connected database and database partition number.

Authorization

- SYSMON authority
- EXECUTE privilege on the SNAP_GET_TBSP_PART_V97 table function.

Example

Retrieve a list of table spaces and their state for the connected database partition of the connected database.

```
SELECT SUBSTR(TBSP_NAME,1,30) AS TBSP_NAME, TBSP_ID,
   SUBSTR(TBSP_STATE,1,30) AS TBSP_STATE
   FROM TABLE(SNAP_GET_TBSP_PART_V97(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
TBSP_NAME                      TBSP_ID              TBSP_STATE
------------------------------ -------------------- ------------...-
SYSCATSPACE                                       0 NORMAL
TEMPSPACE1                                        1 NORMAL
USERSPACE1                                        2 NORMAL
SYSTOOLSPACE                                      3 NORMAL
SYSTOOLSTMPSPACE                                  4 NORMAL

  5 record(s) selected.
```

Information returned

Table 194. Information returned by the SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V97 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR (128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification

Table 194. Information returned by the SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_STATE	VARCHAR (256)	tablespace_state - Table space state. This interface returns a text identifier based on defines in `sqlutil.h` and is combination of the following separated by a '+' sign: • BACKUP_IN_PROGRESS • BACKUP_PENDING • DELETE_PENDING • DISABLE_PENDING • DROP_PENDING • LOAD_IN_PROGRESS • LOAD_PENDING • NORMAL • OFFLINE • PSTAT_CREATION • PSTAT_DELETION • QUIESCED_EXCLUSIVE • QUIESCED_SHARE • QUIESCED_UPDATE • REBAL_IN_PROGRESS • REORG_IN_PROGRESS • RESTORE_IN_PROGRESS • RESTORE_PENDING • ROLLFORWARD_IN_PROGRESS • ROLLFORWARD_PENDING • STORDEF_ALLOWED • STORDEF_CHANGED • STORDEF_FINAL_VERSION • STORDEF_PENDING • SUSPEND_WRITE
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size
TBSP_NUM_QUIESCERS	BIGINT	tablespace_num_quiescers - Number of quiescers
TBSP_STATE_CHANGE_OBJECT_ID	BIGINT	tablespace_state_change_object_id - State change object identification
TBSP_STATE_CHANGE_TBSP_ID	BIGINT	tablespace_state_change_ts_id - State change table space identification
TBSP_MIN_RECOVERY_TIME	TIMESTAMP	tablespace_min_recovery_time - Minimum recovery time for rollforward
TBSP_TOTAL_PAGES	BIGINT	tablespace_total_pages - Total pages in table space

Table 194. Information returned by the SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_USABLE_PAGES	BIGINT	tablespace_usable_pages - Usable pages in table space
TBSP_USED_PAGES	BIGINT	tablespace_used_pages - Used pages in table space
TBSP_FREE_PAGES	BIGINT	tablespace_free_pages - Free pages in table space
TBSP_PENDING_FREE_PAGES	BIGINT	tablespace_pending_free_pages - Pending free pages in table space
TBSP_PAGE_TOP	BIGINT	tablespace_page_top - Table space high water mark
REBALANCER_MODE	VARCHAR (30)	tablespace_rebalancer_mode - Rebalancer mode. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • FWD_REBAL • NO_REBAL • REV_REBAL • FWD_REBAL_OF_2PASS • REV_REBAL_OF_2PASS
REBALANCER_EXTENTS_REMAINING	BIGINT	tablespace_rebalancer_extents_remaining - Total number of extents to be processed by the rebalancer
REBALANCER_EXTENTS_PROCESSED	BIGINT	tablespace_rebalancer_extents_processed - Number of extents the rebalancer has processed
REBALANCER_PRIORITY	BIGINT	tablespace_rebalancer_priority - Current rebalancer priority
REBALANCER_START_TIME	TIMESTAMP	tablespace_rebalancer_start_time - Rebalancer start time
REBALANCER_RESTART_TIME	TIMESTAMP	tablespace_rebalancer_restart_time - Rebalancer restart time
REBALANCER_LAST_EXTENT_MOVED	BIGINT	tablespace_rebalancer_last_extent_moved - Last extent moved by the rebalancer
TBSP_NUM_RANGES	BIGINT	tablespace_num_ranges - Number of ranges in the table space map
TBSP_NUM_CONTAINERS	BIGINT	tablespace_num_containers - Number of containers in table space
TBSP_INITIAL_SIZE	BIGINT	tablespace_initial_size - Initial table space size
TBSP_CURRENT_SIZE	BIGINT	tablespace_current_size - Current table space size
TBSP_MAX_SIZE	BIGINT	tablespace_max_size - Maximum table space size

Table 194. Information returned by the SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V97 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_INCREASE_SIZE	BIGINT	tablespace_increase_size - Increase size in bytes
TBSP_INCREASE_SIZE_PERCENT	SMALLINT	tablespace_increase_size_percent - Increase size by percent
TBSP_LAST_RESIZE_TIME	TIMESTAMP	tablespace_last_resize_time - Time of last successful resize
TBSP_LAST_RESIZE_FAILED	SMALLINT	tablespace_last_resize_failed - Last resize attempt failed
TBSP_PATHS_DROPPED	SMALLINT	Indicates that the table space resides on one or more storage paths that have been dropped (0 - No, 1 - Yes)
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTBSP_QUIESCER administrative view and SNAP_GET_TBSP_QUIESCER table function - Retrieve quiescer table space snapshot information

The SNAPTBSP_QUIESCER administrative view and the SNAP_GET_TBSP_QUIESCER table function return information about quiescers from a table space snapshot.

SNAPTBSP_QUIESCER administrative view

This administrative view allows you to retrieve quiescer table space snapshot information for the currently connected database.

Used with the SNAPTBSP, SNAPTBSP_PART, SNAPTBSP_RANGE, SNAPCONTAINER administrative views, the SNAPTBSP_QUIESCER administrative view provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 195 on page 701 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTBSP_QUIESCER administrative view
- CONTROL privilege on the SNAPTBSP_QUIESCER administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TBSP_QUIESCER table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information on quiesced table spaces for all database partitions for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME, 1, 10) AS TBSP_NAME, QUIESCER_TS_ID,
   QUIESCER_OBJ_ID, QUIESCER_AUTH_ID, QUIESCER_AGENT_ID,
   QUIESCER_STATE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTBSP_QUIESCER ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TBSP_NAME  QUIESCER_TS_ID     QUIESCER_OBJ_ID     QUIESCER_AUTH_ID     ..
---------- --------------...- ---------------...- ----------------...- ..
USERSPACE1                 2                   5 SWALKTY              ..
USERSPACE1                 2                   5 SWALKTY              ..

  2 record(s) selected.
```

Output from this query (continued).

```
... QUIESCER_AGENT_ID    QUIESCER_STATE DBPARTITIONNUM
... -------------------- -------------- --------------
...                    0 EXCLUSIVE                  0
...                65983 EXCLUSIVE                  1
```

Example: Determine the range partitioned table names

If the table is range-partitioned and kept in quiesced state, the different values for table space ID and table ID are represented than in SYSCAT.TABLES. These IDs will appear as the unsigned short representation. In order to find the quiesced table name, you need to find the signed short representation first by calculating the table space ID that is subtracting 65536 (the maximum value) from QEUIESCER_TS_ID and then use this table space ID to locate the quiesced tables. (The actual table space ID can be found in SYSCAT.DATAPARTITIONS for each range partition in the table).

```
SELECT SUBSTR(TBSP_NAME, 1, 10) AS TBSP_NAME,
  CASE WHEN QUIESCER_TS_ID = 65530
    THEN QUIESCER_TS_ID - 65536
    ELSE QUIESCER_TS_ID END as tbspaceid,
  CASE WHEN QUIESCER_TS_ID = 65530
    THEN QUIESCER_OBJ_ID - 65536
    ELSE QUIESCER_OBJ_ID END as tableid
  FROM SYSIBMADM.SNAPTBSP_QUIESCER
  ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TBSP_NAME     TBSPACEID    TABLEID
------------- ------------ --------
TABDATA       -6           -32768
```

```
DATAMART      -6          -32765
SMALL         5           17

  3 record(s) selected.
```

Use the given TBSPACEID and TABLEID provided from above query to find the table schema and name from SYSCAT.TABLES.

```
SELECT CHAR(tabschema, 10)tabschema, CHAR(tabname,15)tabname
  FROM SYSCAT.TABLES
  WHERE tbspaceid = -6 AND tableid in (-32768,-32765)
```

The following is an example of output from this query.

```
TABSCHEMA    TABNAME
------------ --------
TPCD         ORDERS_RP
TPCD         ORDERS_DMART

  2 record(s) selected.
```

```
SELECT CHAR(tabschema, 10)tabschema, CHAR(tabname,15)tabname
  FROM SYSCAT.TABLES
  WHERE tbspaceid = 5 AND tableid = 17
```

The following is an example of output from this query.

```
TABSCHEMA    TABNAME
------------ --------
TPCD         NATION

  1 record(s) selected.
```

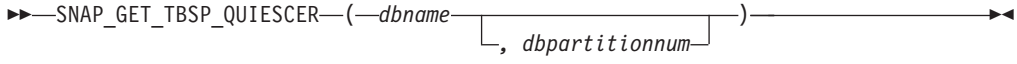
SNAP_GET_TBSP_QUIESCER table function

The SNAP_GET_TBSP_QUIESCER table function returns the same information as the SNAPTBSP_QUIESCER administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TBSP_V91, SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_RANGE, SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_QUIESCER table function provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 195 on page 701 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as

returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_QUIESCER table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TBSP_QUIESCER table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information on quiesced table spaces for database partition 1 for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME, 1, 10) AS TBSP_NAME, QUIESCER_TS_ID,
   QUIESCER_OBJ_ID, QUIESCER_AUTH_ID, QUIESCER_AGENT_ID,
   QUIESCER_STATE, DBPARTITIONNUM
   FROM TABLE( SYSPROC.SNAP_GET_TBSP_QUIESCER( '', 1)) AS T
```

The following is an example of output from this query.

```
TBSP_NAME  QUIESCER_TS_ID     QUIESCER_OBJ_ID     QUIESCER_AUTH_ID     ...
---------- --------------...- ---------------...- ----------------...- ...
USERSPACE1                 2                   5 SWALKTY              ...

  1 record(s) selected.
```

Output from this query (continued).

```
... QUIESCER_AGENT_ID    QUIESCER_STATE DBPARTITIONNUM
... -------------------- -------------- --------------
...                65983 EXCLUSIVE                   1
```

Information returned

Table 195. Information returned by the SNAPTBSP_QUIESCER administrative view and the SNAP_GET_TBSP_QUIESCER table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
QUIESCER_TS_ID	BIGINT	quiescer_ts_id - Quiescer table space identification
QUIESCER_OBJ_ID	BIGINT	quiescer_obj_id - Quiescer object identification
QUIESCER_AUTH_ID	VARCHAR(128)	quiescer_auth_id - Quiescer user authorization identification
QUIESCER_AGENT_ID	BIGINT	quiescer_agent_id - Quiescer agent identification
QUIESCER_STATE	VARCHAR(14)	quiescer_state - Quiescer state. This interface returns a text identifier based on defines in `sqlutil.h` and is one of: • EXCLUSIVE • UPDATE • SHARE
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTBSP_RANGE administrative view and SNAP_GET_TBSP_RANGE table function - Retrieve range snapshot information

The SNAPTBSP_RANGE administrative view and the SNAP_GET_TBSP_RANGE table function return information from a range snapshot.

SNAPTBSP_RANGE administrative view

This administrative view allows you to retrieve range snapshot information for the currently connected database.

Used with the SNAPTBSP, SNAPTBSP_PART, SNAPTBSP_QUIESCER and SNAPCONTAINER administrative views, the SNAPTBSP_RANGE administrative view provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 196 on page 704 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTBSP_RANGE administrative view
- CONTROL privilege on the SNAPTBSP_RANGE administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TBSP_RANGE table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Select information about table space ranges for all database partitions for the currently connected database.

```
SELECT TBSP_ID, SUBSTR(TBSP_NAME, 1, 15) AS TBSP_NAME, RANGE_NUMBER,
   RANGE_STRIPE_SET_NUMBER, RANGE_OFFSET, RANGE_MAX_PAGE,
   RANGE_MAX_EXTENT, RANGE_START_STRIPE, RANGE_END_STRIPE,
   RANGE_ADJUSTMENT, RANGE_NUM_CONTAINER, RANGE_CONTAINER_ID,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPTBSP_RANGE
   ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TBSP_ID     TBSP_NAME       RANGE_NUMBER     RANGE_STRIPE_SET_NUMBER ...
-------...- --------------- ------------...- ----------------------- ...
          0 SYSCATSPACE                    0                       0 ...
          2 USERSPACE1                     0                       0 ...
          3 SYSTOOLSPACE                   0                       0 ...
          2 USERSPACE1                     0                       0 ...
          2 USERSPACE1                     0                       0 ...

  5 record(s) selected.
```

Output from this query (continued).

```
... RANGE_OFFSET     RANGE_MAX_PAGE       RANGE_MAX_EXTENT     ...
... ------------...- -------------------- -------------------- ...
...                0                11515                 2878 ...
...                0                  479                   14 ...
...                0                  251                   62 ...
...                0                  479                   14 ...
...                0                  479                   14 ...
```

Output from this query (continued).

```
... RANGE_START_STRIPE   RANGE_END_STRIPE     RANGE_ADJUSTMENT     ...
... -------------------- -------------------- -------------------- ...
...                    0                 2878                    0 ...
...                    0                   14                    0 ...
...                    0                   62                    0 ...
...                    0                   14                    0 ...
...                    0                   14                    0 ...
```

Output from this query (continued).

```
... RANGE_NUM_CONTAINER  RANGE_CONTAINER_ID   DBPARTITIONNUM
... -------------------- -------------------- --------------
...                    1                    0              0
...                    1                    0              0
...                    1                    0              0
...                    1                    0              1
...                    1                    0              2
```

SNAP_GET_TBSP_RANGE table function

The SNAP_GET_TBSP_RANGE table function returns the same information as the SNAPTBSP_RANGE administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TBSP_V91, SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_QUIESCER and SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_RANGE table function provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 196 on page 704 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_RANGE table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TBSP_RANGE table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Select information on the table space range for the table space with tbsp_id = 2 on the currently connected database partition.

```
SELECT TBSP_ID, SUBSTR(TBSP_NAME, 1, 15) AS TBSP_NAME, RANGE_NUMBER,
   RANGE_STRIPE_SET_NUMBER, RANGE_OFFSET, RANGE_MAX_PAGE, RANGE_MAX_EXTENT,
   RANGE_START_STRIPE, RANGE_END_STRIPE, RANGE_ADJUSTMENT,
   RANGE_NUM_CONTAINER, RANGE_CONTAINER_ID
   FROM TABLE(SNAP_GET_TBSP_RANGE('',-1)) AS T WHERE TBSP_ID = 2
```

The following is an example of output from this query.

```
TBSP_ID     TBSP_NAME       RANGE_NUMBER     ...
-------...- --------------- ------------...- ...
          2 USERSPACE1                     0 ...

  1 record(s) selected.
```

Output from this query (continued).

```
... RANGE_STRIPE_SET_NUMBER RANGE_OFFSET     RANGE_MAX_PAGE        ...
... ----------------------- ------------...- --------------...---- ...
...                       0               0                  3967 ...
```

Output from this query (continued).

```
... RANGE_MAX_EXTENT     RANGE_START_STRIPE   RANGE_END_STRIPE     ...
... -------------------- -------------------- -------------------- ...
...                  123                    0                  123 ...
```

Output from this query (continued).

```
... RANGE_ADJUSTMENT     RANGE_NUM_CONTAINER RANGE_CONTAINER_ID
... -------------------- ------------------- --------------------
...                    0                   1                    0
```

Information returned

Table 196. Information returned by the SNAPTBSP_RANGE administrative view and the SNAP_GET_TBSP_RANGE table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_ID	BIGINT	tablespace_id - Table space identification

Table 196. Information returned by the SNAPTBSP_RANGE administrative view and the SNAP_GET_TBSP_RANGE table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
RANGE_NUMBER	BIGINT	range_number - Range number
RANGE_STRIPE_SET_NUMBER	BIGINT	range_stripe_set_number - Stripe set number
RANGE_OFFSET	BIGINT	range_offset - Range offset
RANGE_MAX_PAGE	BIGINT	range_max_page_number - Maximum page in range
RANGE_MAX_EXTENT	BIGINT	range_max_extent - Maximum extent in range
RANGE_START_STRIPE	BIGINT	range_start_stripe - Start stripe
RANGE_END_STRIPE	BIGINT	range_end_stripe - End stripe
RANGE_ADJUSTMENT	BIGINT	range_adjustment - Range adjustment
RANGE_NUM_CONTAINER	BIGINT	range_num_containers - Number of containers in range
RANGE_CONTAINER_ID	BIGINT	range_container_id - Range container
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPUTIL administrative view and SNAP_GET_UTIL table function - Retrieve utility_info logical data group snapshot information

The SNAPUTIL administrative view and the SNAP_GET_UTIL table function return snapshot information on utilities from the utility_info logical data group.

SNAPUTIL administrative view

Used in conjunction with the SNAPUTIL_PROGRESS administrative view, the SNAPUTIL administrative view provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

The schema is SYSIBMADM.

Refer to Table 197 on page 708 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPUTIL administrative view
- CONTROL privilege on the SNAPUTIL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_UTIL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of utilities and their states on all database partitions for all active databases in the instance that contains the connected database.

```
SELECT UTILITY_TYPE, UTILITY_PRIORITY, SUBSTR(UTILITY_DESCRIPTION, 1, 72)
   AS UTILITY_DESCRIPTION, SUBSTR(UTILITY_DBNAME, 1, 17) AS
   UTILITY_DBNAME, UTILITY_STATE, UTILITY_INVOKER_TYPE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPUTIL ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
UTILITY_TYPE     UTILITY_PRIORITY ...
------------...- ---------------- ...
LOAD                            - ...
LOAD                            - ...
LOAD                            - ...

3 record(s) selected.
```

Output from this query (continued).

```
... UTILITY_DESCRIPTION                                                  ...
... -------------------------------------------------------------------...
... ONLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO TEST .LOADTEST ...
... ONLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO TEST .LOADTEST ...
... ONLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO TEST .LOADTEST ...
```

Output from this query (continued).

```
... UTILITY_DBNAME    UTILITY_STATE UTILITY_INVOKER_TYPE DBPARTITIONNUM
... ----------------- ------------- -------------------- --------------
... SAMPLE            EXECUTE       USER                            0
... SAMPLE            EXECUTE       USER                            1
... SAMPLE            EXECUTE       USER                            2
```

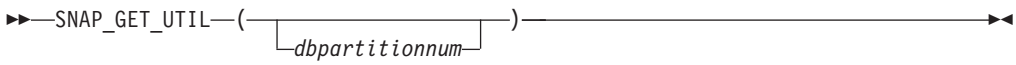
SNAP_GET_UTIL table function

The SNAP_GET_UTIL table function returns the same information as the SNAPUTIL administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_UTIL_PROGRESS table function, the SNAP_GET_UTIL table function provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

Refer to Table 197 on page 708 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_UTIL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_UTIL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of utility ids with their type and state for the currently connected database partition on database SAMPLE.

```
SELECT UTILITY_ID, UTILITY_TYPE, STATE
   FROM TABLE(SNAP_GET_UTIL(-1)) AS T WHERE UTILITY_DBNAME='SAMPLE'
```

The following is an example of output from this query.

```
UTILITY_ID           UTILITY_TYPE               STATE
-------------------- -------------------------- --------
                   1 BACKUP                     EXECUTE

  1 record(s) selected.
```

Information returned

Table 197. Information returned by the SNAPUTIL administrative view and the SNAP_GET_UTIL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
UTILITY_ID	INTEGER	utility_id - Utility ID. Unique to a database partition.
UTILITY_TYPE	VARCHAR(26)	utility_type - Utility type. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • ASYNC_INDEX_CLEANUP • BACKUP • CRASH_RECOVERY • LOAD • REBALANCE • REDISTRIBUTE • REORG • RESTART_RECREATE_INDEX • RESTORE • ROLLFORWARD_RECOVERY • RUNSTATS
UTILITY_PRIORITY	INTEGER	utility_priority - Utility priority. Priority if utility supports throttling, otherwise null.
UTILITY_DESCRIPTION	VARCHAR(2048)	utility_description - Utility description. Can be null.
UTILITY_DBNAME	VARCHAR(128)	utility_dbname - Database operated on by utility
UTILITY_START_TIME	TIMESTAMP	utility_start_time - Utility start time
UTILITY_STATE	VARCHAR(10)	utility_state - Utility state. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • ERROR • EXECUTE • WAIT
UTILITY_INVOKER_TYPE	VARCHAR(10)	utility_invoker_type - Utility invoker type. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTO • USER
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
PROGRESS_LIST_ATTR	VARCHAR(10)	progress_list_attr - Current progress list attributes

Table 197. Information returned by the SNAPUTIL administrative view and the SNAP_GET_UTIL table function (continued)

Column name	Data type	Description or corresponding monitor element
PROGRESS_LIST_CUR_SEQ_NUM	INTEGER	progress_list_current_seq_num - Current® progress list sequence number

SNAPUTIL_PROGRESS administrative view and SNAP_GET_UTIL_PROGRESS table function - Retrieve progress logical data group snapshot information

The SNAPUTIL_PROGRESS administrative view and the SNAP_GET_UTIL_PROGRESS table function return snapshot information about utility progress, in particular, the progress logical data group.

SNAPUTIL_PROGRESS administrative view

Used in conjunction with the SNAPUTIL administrative view, the SNAPUTIL_PROGRESS administrative view provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

The schema is SYSIBMADM.

Refer to Table 198 on page 711 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPUTIL_PROGRESS administrative view
- CONTROL privilege on the SNAPUTIL_PROGRESS administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_UTIL_PROGRESS table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on total and completed units of progress by utility ID.

```
SELECT SELECT UTILITY_ID, PROGRESS_TOTAL_UNITS, PROGRESS_COMPLETED_UNITS,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPUTIL_PROGRESS
```

The following is an example of output from this query.

```
UTILITY_ID PROGRESS_TOTAL_UNITS PROGRESS_COMPLETED_UNITS DBPARTITIONNU
---------- -------------------- ------------------------ ------------
         7                   10                        5            0
         9                   10                        5            1

  1 record(s) selected.
```

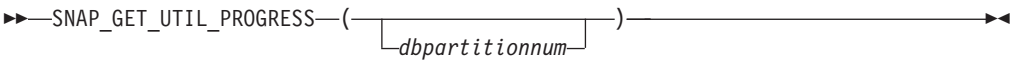
SNAP_GET_UTIL_PROGRESS table function

The SNAP_GET_UTIL_PROGRESS table function returns the same information as the SNAPUTIL_PROGRESS administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_UTIL table function, the SNAP_GET_UTIL_PROGRESS table function provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

Refer to Table 198 on page 711 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_UTIL_PROGRESS table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_UTIL_PROGRESS table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON

- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on the progress of utilities on the currently connect partition.

```
SELECT UTILITY_ID, PROGRESS_TOTAL_UNITS, PROGRESS_COMPLETED_UNITS,
   DBPARTITIONNUM FROM TABLE(SNAP_GET_UTIL_PROGRESS(-1)) as T
```

The following is an example of output from this query.

```
UTILITY_ID PROGRESS_TOTAL_UNITS PROGRESS_COMPLETED_UNITS DBPARTITIONNUM
---------- -------------------- ------------------------ --------------
         7                   10                        5              0

  1 record(s) selected.
```

Information returned

Table 198. Information returned by the SNAPUTIL_PROGRESS administrative view and the SNAP_GET_UTIL_PROGRESS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
UTILITY_ID	INTEGER	utility_id - Utility ID. Unique to a database partition.
PROGRESS_SEQ_NUM	INTEGER	progress_seq_num - Progress sequence number. If serial, the number of the phase. If concurrent, then could be NULL.
UTILITY_STATE	VARCHAR(16)	utility_state - Utility state. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • ERROR • EXECUTE • WAIT
PROGRESS_DESCRIPTION	VARCHAR(2048)	progress_description - Progress description
PROGRESS_START_TIME	TIMESTAMP	progress_start_time - Progress start time. Start time if the phase has started, otherwise NULL.

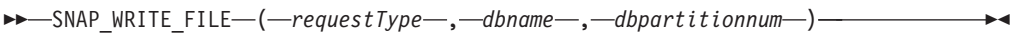
Table 198. Information returned by the SNAPUTIL_PROGRESS administrative view and the SNAP_GET_UTIL_PROGRESS table function (continued)

Column name	Data type	Description or corresponding monitor element
PROGRESS_WORK_METRIC	VARCHAR(16)	progress_work_metric - Progress work metric. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • NOT_SUPPORT • BYTES • EXTENTS • INDEXES • PAGES • ROWS • TABLES
PROGRESS_TOTAL_UNITS	BIGINT	progress_total_units - Total progress work units
PROGRESS_COMPLETED_UNITS	BIGINT	progress_completed_units - Completed progress work units
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAP_WRITE_FILE procedure

The SNAP_WRITE_FILE procedure writes system snapshot data to a file in the `tmp` subdirectory of the instance directory.

Syntax



The schema is SYSPROC.

Procedure parameters

requestType
An input argument of type VARCHAR (32) that specifies a valid snapshot request type. The possible request types are text identifiers based on defines in `sqlmon.h`, and are one of:

- APPL_ALL
- BUFFERPOOLS_ALL
- DB2
- DBASE_ALL
- DBASE_LOCKS
- DBASE_TABLES
- DBASE_TABLESPACES
- DYNAMIC_SQL

dbname
An input argument of type VARCHAR(128) that specifies a valid database

name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If a null value is specified, -1 is set implicitly.

Authorization

To execute the procedure, a user must have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority. The saved snapshot can be read by users who do not have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority by passing null values as the inputs to snapshot table functions.

Example

Take a snapshot of database manager information by specifying a request type of 'DB2' (which corresponds to SQLMA_DB2), and defaulting to the currently connected database and current database partition.

```
CALL SYSPROC.SNAP_WRITE_FILE ('DB2', '', -1)
```

This will result in snapshot data being written to the instance temporary directory, which is sqllib/tmp/SQLMA_DB2.dat on UNIX operating systems, and sqllib\DB2\tmp\SQLMA_DB2.dat on a Windows operating system.

Usage notes

If an unrecognized input parameter is provided, the following error is returned: SQL2032N The "REQUEST_TYPE" parameter is not valid.

SNAPAGENT administrative view and SNAP_GET_AGENT table function – Retrieve agent logical data group application snapshot information

The SNAPAGENT administrative view and the SNAP_GET_AGENT table function return information about agents from an application snapshot, in particular, the agent logical data group.

SNAPAGENT administrative view

This administrative view allows you to retrieve agent logical data group application snapshot information for the currently connected database.

Used with the SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPAPPL_INFO, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAGENT administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 169 on page 576 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPAGENT administrative view
- CONTROL privilege on the SNAPAGENT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_AGENT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve all application snapshot information for the currently connected database from the agent logical data group.

```
SELECT * FROM SYSIBMADM.SNAPAGENT
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         DB_NAME     AGENT_ID        ...
-------------------------- -------...- ---------...--- ...
2005-07-19-11.03.26.740423 SAMPLE                  101 ...
2005-07-19-11.03.26.740423 SAMPLE                   49 ...
                                                       ...
  2 record(s) selected.                                ...
```

Output from this query (continued).

```
...  AGENT_PID            LOCK_TIMEOUT_VAL     DBPARTITIONNUM
...  -------------------- -------------------- --------------
...                 11980                   -1              0
...                 15940                   -1              0
...
...
```

SNAP_GET_AGENT table function

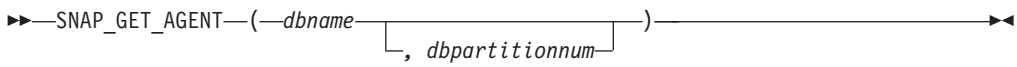
The SNAP_GET_AGENT table function returns the same information as the SNAPAGENT administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_AGENT table function provides information

equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 169 on page 576 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_AGENT table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_AGENT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve all application snapshot information for all applications in all active databases.

```
SELECT * FROM TABLE(SNAP_GET_AGENT(CAST(NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         DB_NAME     AGENT_ID      ...
-------------------------- -------...- --------...-- ...
2006-01-03-17.21.38.530785 SAMPLE                48 ...
2006-01-03-17.21.38.530785 SAMPLE                47 ...
2006-01-03-17.21.38.530785 SAMPLE                46 ...
2006-01-03-17.21.38.530785 TESTDB                30 ...
2006-01-03-17.21.38.530785 TESTDB                29 ...
2006-01-03-17.21.38.530785 TESTDB                28 ...

6 record(s) selected.
```

Output from this query (continued).

```
... AGENT_PID        LOCK_TIMEOUT_VAL     DBPARTITIONNUM
... ---------...---- -------------------- --------------
...             7696                   -1              0
...             8536                   -1              0
...             6672                   -1              0
...             2332                   -1              0
...             8360                   -1              0
...             6736                   -1              0
...
```

Information returned

Table 199. Information returned by the SNAPAGENT administrative view and the SNAP_GET_AGENT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
AGENT_PID	BIGINT	agent_pid - Engine dispatchable unit (EDU)
LOCK_TIMEOUT_VAL	BIGINT	lock_timeout_val - Lock timeout (seconds)
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.

SNAPAGENT_MEMORY_POOL administrative view and SNAP_GET_AGENT_MEMORY_POOL table function – Retrieve memory_pool logical data group snapshot information

The SNAPAGENT_MEMORY_POOL administrative view and the SNAP_GET_AGENT_MEMORY_POOL table function return information about memory usage at the agent level.

SNAPAGENT_MEMORY_POOL administrative view

This administrative view allows you to retrieve the memory_pool logical data group snapshot information about memory usage at the agent level for the currently connected database.

Used with the SNAPAGENT, SNAPAPPL, SNAPAPPL_INFO, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAGENT_MEMORY_POOL administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 170 on page 579 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPAGENT_MEMORY_POOL administrative view
- CONTROL privilege on the SNAPAGENT_MEMORY_POOL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_AGENT_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size.

```
SELECT AGENT_ID, POOL_ID, POOL_CUR_SIZE FROM SYSIBMADM.SNAPAGENT_MEMORY_POOL
```

The following is an example of output from this query.

```
AGENT_ID       POOL_ID POOL_  CUR_SIZE
--------...--- ------------- ---------...------
            48 APPLICATION                65536
            48 OTHER                      65536
            48 APPL_CONTROL               65536
            47 APPLICATION                65536
            47 OTHER                     131072
            47 APPL_CONTROL               65536
            46 OTHER                     327680
            46 APPLICATION               262144
            46 APPL_CONTROL               65536

9 record(s) selected.
```

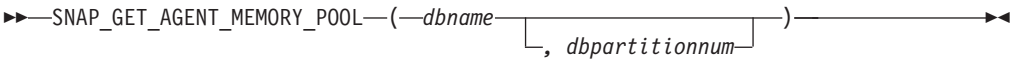
SNAP_GET_AGENT_MEMORY_POOL table function

The SNAP_GET_AGENT_MEMORY_POOL table function returns the same information as the SNAPAGENT_MEMORY_POOL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_AGENT_MEMORY_POOL table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command.

Refer to Table 170 on page 579 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_AGENT_MEMORY_POOL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_AGENT_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size for all databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID, POOL_ID, POOL_CUR_SIZE
   FROM TABLE(SNAP_GET_AGENT_MEMORY_POOL(CAST (NULL AS VARCHAR(128)), -1))
   AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID       POOL_ID        POOL_CUR_SIZE
-------- --------...--- -------------- --------------------
SAMPLE               48 APPLICATION                   65536
SAMPLE               48 OTHER                         65536
SAMPLE               48 APPL_CONTROL                  65536
SAMPLE               47 APPLICATION                   65536
SAMPLE               47 OTHER                        131072
SAMPLE               47 APPL_CONTROL                  65536
SAMPLE               46 OTHER                        327680
SAMPLE               46 APPLICATION                  262144
SAMPLE               46 APPL_CONTROL                  65536
TESTDB               30 APPLICATION                   65536
TESTDB               30 OTHER                         65536
TESTDB               30 APPL_CONTROL                  65536
TESTDB               29 APPLICATION                   65536
TESTDB               29 OTHER                        131072
TESTDB               29 APPL_CONTROL                  65536
TESTDB               28 OTHER                        327680
TESTDB               28 APPLICATION                   65536
TESTDB               28 APPL_CONTROL                  65536

18 record(s) selected.
```

Information returned

Table 200. Information returned by the SNAPAGENT_MEMORY_POOL administrative view and the SNAP_GET_AGENT_MEMORY_POOL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
AGENT_PID	BIGINT	agent_pid - Engine dispatchable unit (EDU)

Table 200. Information returned by the SNAPAGENT_MEMORY_POOL administrative view and the SNAP_GET_AGENT_MEMORY_POOL table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ID	VARCHAR(14)	pool_id - Memory pool identifier. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • APP_GROUP • APPL_CONTROL • APPLICATION • BP • CAT_CACHE • DATABASE • DFM • FCMBP • IMPORT_POOL • LOCK_MGR • MONITOR • OTHER • PACKAGE_CACHE • QUERY • SHARED_SORT • SORT • STATEMENT • STATISTICS • UTILITY
POOL_CUR_SIZE	BIGINT	pool_cur_size - Current size of memory pool
POOL_WATERMARK	BIGINT	pool_watermark - Memory pool watermark
POOL_CONFIG_SIZE	BIGINT	pool_config_size - Configured size of memory pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPAPPL_INFO administrative view and SNAP_GET_APPL_INFO_V95 table function - Retrieve appl_info logical data group snapshot information

The SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function return information about applications from an application snapshot, in particular, the appl_info logical data group.

SNAPAPPL_INFO administrative view

This administrative view allows you to retrieve appl_info logical data group snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAPPL_INFO administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 171 on page 583 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPAPPL_INFO administrative view
- CONTROL privilege on the SNAPAPPL_INFO administrative view
- DATAACCESS authority

Additionally, one of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_APPL_INFO_V95 table function
- DATAACCESS authority

Also, one of the following authorities is required:

- SYSMON
- SYSMAINT
- SYSCTRL
- SYSADM

Example

Retrieve the status of the applications connected to the current database.

```
SELECT AGENT_ID, SUBSTR(APPL_NAME,1,10) AS APPL_NAME, APPL_STATUS
   FROM SYSIBMADM.SNAPAPPL_INFO
```

The following is an example of output from this query.

```
AGENT_ID             APPL_NAME  APPL_STATUS
-------------------- ---------- ----------------------
                 101 db2bp.exe  UOWEXEC
                  49 db2bp.exe  CONNECTED

  2 record(s) selected.
```

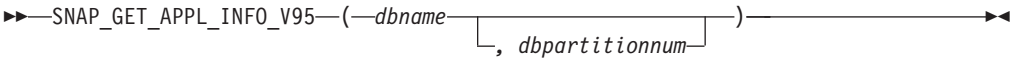
SNAP_GET_APPL_INFO_V95 table function

The SNAP_GET_APPL_INFO_V95 table function returns the same information as the SNAPAPPL_INFO administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_APPL_INFO_V95 table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 171 on page 583 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_APPL_INFO_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_APPL_INFO_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve the status of all applications on the connected database partition.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID,
   SUBSTR(APPL_NAME,1,10) AS APPL_NAME, APPL_STATUS
   FROM TABLE(SNAP_GET_APPL_INFO_V95(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID             APPL_NAME  APPL_STATUS
-------- -------------------- ---------- ----------------------
TOOLSDB                    14 db2bp.exe  CONNECTED
SAMPLE                     15 db2bp.exe  UOWEXEC
SAMPLE                      8 javaw.exe  CONNECTED
SAMPLE                      7 db2bp.exe  UOWWAIT

  4 record(s) selected.
```

The following shows what you obtain when you SELECT from the result of the table function.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AUTHORITY_LVL
   FROM TABLE(SNAP_GET_APPL_INFO_V95(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AUTHORITY_LVL
-------- -----------------------------------------------------....
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)

3 record(s) selected.
```

Information returned

Table 201. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)

Table 201. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
APPL_STATUS	VARCHAR(22)	appl_status - Application status. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • BACKUP • COMMIT_ACT • COMP • CONNECTED • CONNECTPEND • CREATE_DB • DECOUPLED • DISCONNECTPEND • INTR • IOERROR_WAIT • LOAD • LOCKWAIT • QUIESCE_TABLESPACE • RECOMP • REMOTE_RQST • RESTART • RESTORE • ROLLBACK_ACT • ROLLBACK_TO_SAVEPOINT • TEND • THABRT • THCOMT • TPREP • UNLOAD • UOWEXEC • UOWWAIT • WAITFOR_REMOTE
CODEPAGE_ID	BIGINT	codepage_id - ID of code page used by application
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
COORD_NODE_NUM	SMALLINT	coord_node - Coordinating node

Table 201. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
AUTHORITY_LVL	VARCHAR(512)	authority_bitmap - User authorization level. This interface returns a text identifier based on the database authorities defined in sql.h and their source, and has the following format: authority(source, ...) + authority(source, ...) + ... The source of an authority can be multiple: either from a USER, a GROUP, or a USER and a GROUP. Possible values for "authority": • ACCESSCTRL • BINDADD • CONNECT • CREATE_EXT_RT • CREATE_NOT_FENC • CREATETAB • DATAACCESS • DBADM • EXPLAIN • IMPLICIT_SCHEMA • LOAD • LIBADM • QUIESCE_CONN • SECADM • SQLADM • SYSADM • SYSCTRL • SYSMAINT • SYSMON • SYSQUIESCE • WLMADM Possible values for "source": • USER – authority granted to the user or to a role granted to the user. • GROUP – authority granted to a group to which the user belongs or to a role granted to the group to which the user belongs.
CLIENT_PID	BIGINT	client_pid - Client process ID
COORD_AGENT_PID	BIGINT	coord_agent_pid - Coordinator agent

Table 201. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
STATUS_CHANGE_TIME	TIMESTAMP	status_change_time - Application status change time
CLIENT_PLATFORM	VARCHAR(12)	client_platform - Client operating platform. This interface returns a text identifier based on the defines in `sqlmon.h`, • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS

Table 201. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
CLIENT_PROTOCOL	VARCHAR(10)	client_protocol - Client communication protocol. This interface returns a text identifier based on the defines in `sqlmon.h`, • CPIC • LOCAL • NETBIOS • NPIPE • TCPIP (for DB2 UDB) • TCPIP4 • TCPIP6
TERRITORY_CODE	SMALLINT	territory_code - Database territory code
APPL_NAME	VARCHAR(256)	appl_name - Application name
APPL_ID	VARCHAR(128)	appl_id - Application ID
SEQUENCE_NO	VARCHAR(4)	sequence_no - Sequence number
PRIMARY_AUTH_ID	VARCHAR(128)	auth_id - Authorization ID
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
CLIENT_NNAME	VARCHAR(128)	The client_nname monitor element is deprecated. The value returned is not a valid value.
CLIENT_PRDID	VARCHAR(128)	client_prdid - Client product/version ID
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
CLIENT_DB_ALIAS	VARCHAR(128)	client_db_alias - Database alias used by application
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
EXECUTION_ID	VARCHAR(128)	execution_id - User login ID
CORR_TOKEN	VARCHAR(128)	corr_token - DRDA correlation token
TPMON_CLIENT_USERID	VARCHAR(256)	tpmon_client_userid - TP monitor client user ID
TPMON_CLIENT_WKSTN	VARCHAR(256)	tpmon_client_wkstn - TP monitor client workstation name
TPMON_CLIENT_APP	VARCHAR(256)	tpmon_client_app - TP monitor client application name
TPMON_ACC_STR	VARCHAR(200)	tpmon_acc_str - TP monitor client accounting string
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.
WORKLOAD_ID	INTEGER	Current workload ID.

Table 201. Information returned by the SNAPAPPL_INFO administrative view and the SNAP_GET_APPL_INFO_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
IS_SYSTEM_APPL	SMALLINT	The value of IS_SYSTEM_APPL indicates whether or not the application is a DB2 internal system application: 0 means it is a user application 1 means it is a system application. An example of a DB2 system application is a DB2 event monitor. In general, the names of DB2 system applications begin with "db2". For example: db2stmm, db2taskd.

SNAPAPPL administrative view and SNAP_GET_APPL_V95 table function - Retrieve appl logical data group snapshot information

The "SNAPAPPL administrative view" on page 588 and the "SNAP_GET_APPL_V95 table function" on page 589 return information about applications from an application snapshot, in particular, the appl logical data group.

SNAPAPPL administrative view

This administrative view allows you to retrieve appl logical data group snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL_INFO, SNAPSTMT and SNAPSUBSECTION administrative views, the SNAPAPPL administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS ON database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 172 on page 591 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPAPPL administrative view
- CONTROL privilege on the SNAPAPPL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_APPL_V95 table function

- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on rows read and written for each application in the connected database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID, ROWS_READ, ROWS_WRITTEN
   FROM SYSIBMADM.SNAPAPPL
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID             ROWS_READ            ROWS_WRITTEN
-------- -------------------- -------------------- --------------------
SAMPLE                     7                   25                    0

  1 record(s) selected.
```

SNAP_GET_APPL_V95 table function

The SNAP_GET_APPL_V95 table function returns the same information as the SNAPAPPL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_INFO_V95, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_APPL_V95 table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 172 on page 591 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty

string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_APPL_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_APPL_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on rows read and written for each application for all active databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID, ROWS_READ, ROWS_WRITTEN
   FROM TABLE (SNAP_GET_APPL_V95(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID       ROWS_READ      ROWS_WRITTEN
-------- --------...--- ---------...-- ------------...-
WSDB                679              0                 0
WSDB                461              3                 0
WSDB                460              4                 0
TEST                680              4                 0
TEST                455              6                 0
TEST                454              0                 0
TEST                453             50                 0
```

Information returned

Table 202. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
UOW_LOG_SPACE_USED	BIGINT	uow_log_space_used - Unit of work log space used
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
INACT_STMTHIST_SZ	BIGINT	stmt_history_list_size - Statement history list size
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes

Table 202. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
TOTAL_SORTS	BIGINT	total_sorts - Total sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
COMMIT_SQL_STMTS	BIGINT	commit_sql_stmts - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	rollback_sql_stmts - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	dynamic_sql_stmts - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	static_sql_stmts - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	failed_sql_stmts - Failed statement operations
SELECT_SQL_STMTS	BIGINT	select_sql_stmts - Select SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements

Table 202. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
UID_SQL_STMTS	BIGINT	uid_sql_stmts - UPDATE/INSERT/DELETE SQL statements executed
INT_AUTO_REBINDS	BIGINT	int_auto_rebinds - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
INT_COMMITS	BIGINT	int_commits - Internal commits
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	int_deadlock_rollbacks - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_SELECTED	BIGINT	rows_selected - Rows selected
BINDS_PRECOMPILES	BIGINT	binds_precompiles - Binds/precompiles attempted
OPEN_REM_CURS	BIGINT	open_rem_curs - Open remote cursors
OPEN_REM_CURS_BLK	BIGINT	open_rem_curs_blk - Open remote cursors with blocking
REJ_CURS_BLK	BIGINT	rej_curs_blk - Rejected block cursor requests
ACC_CURS_BLK	BIGINT	acc_curs_blk - Accepted block cursor requests
SQL_REQS_SINCE_COMMIT	BIGINT	sql_reqs_since_commit - SQL requests since last commit
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
OPEN_LOC_CURS	BIGINT	open_loc_curs - Open local cursors
OPEN_LOC_CURS_BLK	BIGINT	open_loc_curs_blk - Open local cursors with blocking
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts

Table 202. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
CAT_CACHE_OVERFLOWS	BIGINT	cat_cache_overflows - Catalog cache overflows
NUM_AGENTS	BIGINT	num_agents - Number of agents working on a statement
AGENTS_STOLEN	BIGINT	agents_stolen - Stolen agents
ASSOCIATED_AGENTS_TOP	BIGINT	associated_agents_top - Maximum number of associated agents
APPL_PRIORITY	BIGINT	appl_priority - Application agent priority
APPL_PRIORITY_TYPE	VARCHAR(16)	appl_priority_type - Application priority type. This interface returns a text identifier, based on defines in `sqlmon.h`, and is one of: • DYNAMIC_PRIORITY • FIXED_PRIORITY
PREFETCH_WAIT_TIME	BIGINT	prefetch_wait_time - Time waited for prefetch
APPL_SECTION_LOOKUPS	BIGINT	appl_section_lookups - Section lookups
APPL_SECTION_INSERTS	BIGINT	appl_section_inserts - Section inserts
LOCKS_WAITING	BIGINT	locks_waiting - Current agents waiting on locks
TOTAL_HASH_JOINS	BIGINT	total_hash_joins - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	hash_join_overflows - Hash join overflows
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	hash_join_small_overflows - Hash join small overflows
APPL_IDLE_TIME	BIGINT	appl_idle_time - Application idle time
UOW_LOCK_WAIT_TIME	BIGINT	uow_lock_wait_time - Total time unit of work waited on locks
UOW_COMP_STATUS	VARCHAR(14)	uow_comp_status - Unit of work completion status. This interface returns a text identifier, based on defines in `sqlmon.h`, and is one of: • APPL_END • UOWABEND • UOWCOMMIT • UOWDEADLOCK • UOWLOCKTIMEOUT • UOWROLLBACK • UOWUNKNOWN

Table 202. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
AGENT_USR_CPU_TIME_S	BIGINT	agent_usr_cpu_time - User CPU time used by agent (in seconds)*
AGENT_USR_CPU_TIME_MS	BIGINT	agent_usr_cpu_time - User CPU time used by agent (fractional, in microseconds)*
AGENT_SYS_CPU_TIME_S	BIGINT	agent_sys_cpu_time - System CPU time used by agent (in seconds)*
AGENT_SYS_CPU_TIME_MS	BIGINT	agent_sys_cpu_time - System CPU time used by agent (fractional, in microseconds)*
APPL_CON_TIME	TIMESTAMP	appl_con_time - Connection request start timestamp
CONN_COMPLETE_TIME	TIMESTAMP	conn_complete_time - Connection request completion timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
UOW_START_TIME	TIMESTAMP	uow_start_time - Unit of work start timestamp
UOW_STOP_TIME	TIMESTAMP	uow_stop_time - Unit of work stop timestamp
PREV_UOW_STOP_TIME	TIMESTAMP	prev_uow_stop_time - Previous unit of work completion timestamp
UOW_ELAPSED_TIME_S	BIGINT	uow_elapsed_time - Most recent unit of work elapsed time (in seconds)*
UOW_ELAPSED_TIME_MS	BIGINT	uow_elapsed_time - Most recent unit of work elapsed time (fractional, in microseconds)*
ELAPSED_EXEC_TIME_S	BIGINT	elapsed_exec_time - Statement execution elapsed time (in seconds)*
ELAPSED_EXEC_TIME_MS	BIGINT	elapsed_exec_time - Statement execution elapsed time (fractional, in microseconds)*
INBOUND_COMM_ADDRESS	VARCHAR(32)	inbound_comm_address - Inbound communication address
LOCK_TIMEOUT_VAL	BIGINT	lock_timeout_val - Lock timeout (seconds)
PRIV_WORKSPACE_NUM_ OVERFLOWS	BIGINT	priv_workspace_num_overflows - Private workspace overflows
PRIV_WORKSPACE_SECTION_ INSERTS	BIGINT	priv_workspace_section_inserts - Private workspace section inserts
PRIV_WORKSPACE_SECTION_ LOOKUPS	BIGINT	priv_workspace_section_lookups - Private workspace section lookups

Table 202. Information returned by the SNAPAPPL administrative view and the SNAP_GET_APPL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
PRIV_WORKSPACE_SIZE_TOP	BIGINT	priv_workspace_size_top - Maximum private workspace size
SHR_WORKSPACE_NUM_OVERFLOWS	BIGINT	shr_workspace_num_overflows - Shared workspace overflows
SHR_WORKSPACE_SECTION_INSERTS	BIGINT	shr_workspace_section_inserts - Shared workspace section inserts
SHR_WORKSPACE_SECTION_LOOKUPS	BIGINT	shr_workspace_section_lookups - Shared workspace section lookups
SHR_WORKSPACE_SIZE_TOP	BIGINT	shr_workspace_size_top - Maximum shared workspace size
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.
CAT_CACHE_SIZE_TOP	BIGINT	cat_cache_size_top - Catalog cache high water mark
TOTAL_OLAP_FUNCS	BIGINT	The total number of OLAP functions executed.
OLAP_FUNC_OVERFLOWS	BIGINT	The number of times that OLAP function data exceeded the available sort heap space.
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPBP administrative view and SNAP_GET_BP_V95 table function - Retrieve bufferpool logical group snapshot information

The SNAPBP administrative view and the SNAP_GET_BP_V95 table function return information about buffer pools from a bufferpool snapshot, in particular, the bufferpool logical data group.

SNAPBP administrative view

This administrative view allows you to retrieve bufferpool logical group snapshot information for the currently connected database.

Used with the SNAPBP_PART administrative view, the SNAPBP administrative view provides the data equivalent to the GET SNAPSHOT FOR BUFFERPOOLS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 173 on page 599 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPBP administrative view
- CONTROL privilege on the SNAPBP administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_BP_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve data and index writes for all the bufferpools of the currently connected database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME,SUBSTR(BP_NAME,1,15)
   AS BP_NAME,POOL_DATA_WRITES,POOL_INDEX_WRITES
   FROM SYSIBMADM.SNAPBP
```

The following is an example of output from this query.

```
DB_NAME  BP_NAME         POOL_DATA_WRITES     POOL_INDEX_WRITES
-------- --------------- -------------------- --------------------
TEST     IBMDEFAULTBP                       0                    0
TEST     IBMSYSTEMBP4K                      0                    0
TEST     IBMSYSTEMBP8K                      0                    0
TEST     IBMSYSTEMBP16K                     0                    0
TEST     IBMSYSTEMBP32K                     0                    0

5 record(s) selected
```

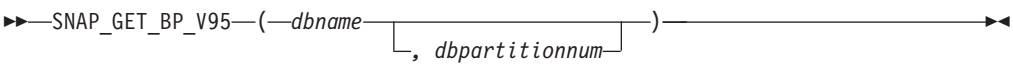
SNAP_GET_BP_V95 table function

The SNAP_GET_BP_V95 table function returns the same information as the SNAPBP administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_BP_PART table function, the SNAP_GET_BP_V95 table function provides the data equivalent to the GET SNAPSHOT FOR ALL BUFFERPOOLS CLP command.

Refer to Table 173 on page 599 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_BP_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_BP_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve total physical and logical reads for all bufferpools for all active databases for the currently connected database partition.

```
SELECT SUBSTR(T.DB_NAME,1,10) AS DB_NAME,
   SUBSTR(T.BP_NAME,1,20) AS BP_NAME,
   (T.POOL_DATA_L_READS+T.POOL_INDEX_L_READS) AS TOTAL_LOGICAL_READS,
```

```
(T.POOL_DATA_P_READS+T.POOL_INDEX_P_READS) AS TOTAL_PHYSICAL_READS,
T.DBPARTITIONNUM
FROM TABLE(SNAP_GET_BP_V95(CAST(NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
DB_NAME    BP_NAME          TOTAL_LOGICAL_READS  ...
---------- ------------...- -------------------- ...
SAMPLE     IBMDEFAULTBP                        0 ...
TOOLSDB    IBMDEFAULTBP                        0 ...
TOOLSDB    BP32K0000                           0 ...

  3 record(s) selected.
```

Output from this query (continued).

```
... TOTAL_PHYSICAL_READS DBPARTITIONNUM
... -------------------- --------------
...                    0              0
...                    0              0
...                    0              0
```

Information returned

Table 203. Information returned by the SNAPBP administrative view and the SNAP_GET_BP_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time

Table 203. Information returned by the SNAPBP administrative view and the SNAP_GET_BP_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
FILES_CLOSED	BIGINT	files_closed - Database files closed
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads

Table 203. Information returned by the SNAPBP administrative view and the SNAP_GET_BP_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
PAGES_FROM_BLOCK_IOS	BIGINT	pages_from_block_ios - Total number of pages read by block I/O
PAGES_FROM_VECTORED_IOS	BIGINT	pages_from_vectored_ios - Total pages read by vectored I/O
VECTORED_IOS	BIGINT	vectored_ios - Number of vectored I/O requests
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPBP_PART administrative view and SNAP_GET_BP_PART table function – Retrieve bufferpool_nodeinfo logical data group snapshot information

The SNAPBP_PART administrative view and the SNAP_GET_BP_PART table function return information about buffer pools from a bufferpool snapshot, in particular, the bufferpool_nodeinfo logical data group.

SNAPBP_PART administrative view

This administrative view allows you to retrieve bufferpool_nodeinfo logical data group snapshot information for the currently connected database.

Used with the SNAPBP administrative view, the SNAPBP_PART administrative view provides the data equivalent to the GET SNAPSHOT FOR BUFFERPOOLS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 174 on page 604 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:
- SELECT privilege on the SNAPBP_PART administrative view
- CONTROL privilege on the SNAPBP_PART administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the SNAP_GET_BP_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve data for all bufferpools when connected to SAMPLE database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(BP_NAME,1,15) AS BP_NAME,
   BP_CUR_BUFFSZ, BP_NEW_BUFFSZ, BP_PAGES_LEFT_TO_REMOVE, BP_TBSP_USE_COUNT
   FROM SYSIBMADM.SNAPBP_PART
```

The following is an example of output from this query.

```
DB_NAME  BP_NAME         BP_CUR_BUFFSZ        BP_NEW_BUFFSZ        ...
-------- --------------- -------------------- -------------------- ...
SAMPLE   IBMDEFAULTBP                   1000                 1000 ...
SAMPLE   IBMSYSTEMBP4K                    16                   16 ...
SAMPLE   IBMSYSTEMBP8K                    16                   16 ...
SAMPLE   IBMSYSTEMBP16K                   16                   16 ...
                                                                  ...
  4 record(s) selected.
```

Output from this query (continued).

```
... BP_PAGES_LEFT_TO_REMOVE BP_TBSP_USE_COUNT
... ----------------------- --------------------
...                       0                    3
...                       0                    0
...                       0                    0
...                       0                    0
...
```

SNAP_GET_BP_PART table function

The SNAP_GET_BP_PART table function returns the same information as the SNAPBP_PART administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_BP_V95 table function, the SNAP_GET_BP_PART table function provides the data equivalent to the GET SNAPSHOT FOR ALL BUFFERPOOLS CLP command.

Refer to Table 174 on page 604 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot for all bufferpools in all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_BP_PART table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_BP_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve data for all bufferpools for all active databases when connected to the SAMPLE database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(BP_NAME,1,15) AS BP_NAME,
   BP_CUR_BUFFSZ, BP_NEW_BUFFSZ, BP_PAGES_LEFT_TO_REMOVE, BP_TBSP_USE_COUNT
   FROM TABLE(SNAP_GET_BP_PART(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  BP_NAME         BP_CUR_BUFFSZ        BP_NEW_BUFFSZ        ...
-------- -------------- -------------------- -------------------- ...
SAMPLE   IBMDEFAULTBP                   250                  250 ...
SAMPLE   IBMSYSTEMBP4K                   16                   16 ...
SAMPLE   IBMSYSTEMBP8K                   16                   16 ...
SAMPLE   IBMSYSTEMBP16K                  16                   16 ...
SAMPLE   IBMSYSTEMBP32K                  16                   16 ...
TESTDB   IBMDEFAULTBP                   250                  250 ...
TESTDB   IBMSYSTEMBP4K                   16                   16 ...
TESTDB   IBMSYSTEMBP8K                   16                   16 ...
TESTDB   IBMSYSTEMBP16K                  16                   16 ...
TESTDB   IBMSYSTEMBP32K                  16                   16 ...

...
```

Output from this query (continued).

```
... BP_PAGES_LEFT_TO_REMOVE BP_TBSP_USE_COUNT
... ----------------------- --------------------
...                       0                    3
...                       0                    0
...                       0                    0
...                       0                    0
...                       0                    0
...                       0                    3
...                       0                    0
...                       0                    0
...                       0                    0
...                       0                    0

...
```

Information returned

Table 204. Information returned by the SNAPBP_PART administrative view and the SNAP_GET_BP_PART table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
DB_NAME	VARCHAR(128)	db_name - Database name
BP_CUR_BUFFSZ	BIGINT	bp_cur_buffsz - current size of buffer pool
BP_NEW_BUFFSZ	BIGINT	bp_new_buffsz - New buffer pool size
BP_PAGES_LEFT_TO_REMOVE	BIGINT	bp_pages_left_to_remove - Number of pages left to remove
BP_TBSP_USE_COUNT	BIGINT	bp_tbsp_use_count - Number of table spaces mapped to buffer pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPCONTAINER administrative view and SNAP_GET_CONTAINER_V91 table function - Retrieve tablespace_container logical data group snapshot information

The SNAPCONTAINER administrative view and the SNAP_GET_CONTAINER_V91 table function return table space snapshot information from the tablespace_container logical data group.

SNAPCONTAINER administrative view

This administrative view allows you to retrieve tablespace_container logical data group snapshot information for the currently connected database.

Used with the SNAPTBSP, SNAPTBSP_PART, SNAPTBSP_QUIESCER and SNAPTBSP_RANGE administrative views, the SNAPCONTAINER administrative view returns data equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 175 on page 608 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPCONTAINER administrative view
- CONTROL privilege on the SNAPCONTAINER administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_CONTAINER_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details for the table space containers for all database partitions for the currently connected database.

```
SELECT SNAPSHOT_TIMESTAMP, SUBSTR(TBSP_NAME, 1, 15) AS TBSP_NAME,
   TBSP_ID, SUBSTR(CONTAINER_NAME, 1, 20) AS CONTAINER_NAME,
   CONTAINER_ID, CONTAINER_TYPE, ACCESSIBLE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPCONTAINER ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         TBSP_NAME       TBSP_ID     ...
-------------------------- --------------- -------...- ...
2006-01-08-16.49.24.639945 SYSCATSPACE              0 ...
2006-01-08-16.49.24.639945 TEMPSPACE1               1 ...
2006-01-08-16.49.24.639945 USERSPACE1               2 ...
2006-01-08-16.49.24.639945 SYSTOOLSPACE             3 ...
2006-01-08-16.49.24.640747 TEMPSPACE1               1 ...
2006-01-08-16.49.24.640747 USERSPACE1               2 ...
2006-01-08-16.49.24.639981 TEMPSPACE1               1 ...
2006-01-08-16.49.24.639981 USERSPACE1               2 ...
                                                      ...
8 record(s) selected.
```

Output from this query (continued).

```
... CONTAINER_NAME       CONTAINER_ID     CONTAINER_TYPE   ...
... -------------------- ------------...- ---------------- ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 PATH             ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 PATH             ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
... /home/swalkty/swalkt                0 PATH             ...
... /home/swalkty/swalkt                0 FILE_EXTENT_TAG  ...
```

Output from this query (continued).

```
... ACCESSIBLE DBPARTITIONNUM
... ---------- --------------
...          1              0
...          1              0
...          1              0
...          1              0
...          1              1
...          1              1
...          1              2
...          1              2
```

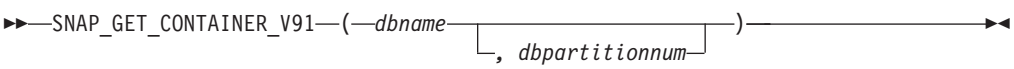
SNAP_GET_CONTAINER_V91 table function

The SNAP_GET_CONTAINER_V91 table function returns the same information as the SNAPCONTAINER administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TBSP_V91, SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_QUIESCER and SNAP_GET_TBSP_RANGE table functions, the SNAP_GET_CONTAINER_V91 table function returns data equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 175 on page 608 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_CONTAINER_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_CONTAINER_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details for the table space containers on the currently connected database on the currently connected database partition.

```
SELECT SNAPSHOT_TIMESTAMP, TBSP_NAME, TBSP_ID, CONTAINER_NAME,
   CONTAINER_ID, CONTAINER_TYPE, ACCESSIBLE
   FROM TABLE(SNAP_GET_CONTAINER_V91('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         TBSP_NAME            TBSP_ID ...
-------------------------- -------------------- ------- ...
2005-04-25-14.42.10.899253 SYSCATSPACE                0 ...
2005-04-25-14.42.10.899253 TEMPSPACE1                 1 ...
2005-04-25-14.42.10.899253 USERSPACE1                 2 ...
2005-04-25-14.42.10.899253 SYSTOOLSPACE               3 ...
2005-04-25-14.42.10.899253 MYTEMP                     4 ...
2005-04-25-14.42.10.899253 WHATSNEWTEMPSPACE          5 ...
```

Output from this query (continued).

```
... CONTAINER_NAME                              CONTAINER_ID ...
... ------------------------------------------- ------------ ...
... D:\DB2\NODE0000\SQL00002\SQLT0000.0                    0 ...
... D:\DB2\NODE0000\SQL00002\SQLT0001.0                    0 ...
... D:\DB2\NODE0000\SQL00002\SQLT0002.0                    0 ...
... D:\DB2\NODE0000\SQL00002\SYSTOOLSPACE                  0 ...
... D:\DB2\NODE0000\SQL003                                 0 ...
... d:\DGTTsWhatsNewContainer                              0 ...
```

Output from this query (continued).

```
... CONTAINER_TYPE ACCESSIBLE
... -------------- ----------
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
... CONT_PATH               1
```

Information returned

NOTE: The BUFFERPOOL database manager monitor switch must be turned on in order for the file system information to be returned.

Table 205. Information returned by the SNAPCONTAINER administrative view and the SNAP_GET_CONTAINER_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
CONTAINER_NAME	VARCHAR(256)	container_name - Container name
CONTAINER_ID	BIGINT	container_id - Container identification
CONTAINER_TYPE	VARCHAR(16)	container_type - Container type. This is a text identifier based on the defines in `sqlutil.h` and is one of: • DISK_EXTENT_TAG • DISK_PAGE_TAG • FILE_EXTENT_TAG • FILE_PAGE_TAG • PATH
TOTAL_PAGES	BIGINT	container_total_pages - Total pages in container
USABLE_PAGES	BIGINT	container_usable_pages - Usable pages in container
ACCESSIBLE	SMALLINT	container_accessible - Accessibility of container
STRIPE_SET	BIGINT	container_stripe_set - Stripe set

Table 205. Information returned by the SNAPCONTAINER administrative view and the SNAP_GET_CONTAINER_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
FS_ID	VARCHAR(22)	fs_id - Unique file system identification number
FS_TOTAL_SIZE	BIGINT	fs_total_size - Total size of a file system
FS_USED_SIZE	BIGINT	fs_used_size - Amount of space used on a file system

SNAPDB administrative view and SNAP_GET_DB_V95 table function - Retrieve snapshot information from the dbase logical group

Note: The SNAP_GET_DB_V95 table function has been deprecated and replaced by the SNAP_GET_DB_V97 table function - Retrieve snapshot information from the dbase logical group..

The "SNAPDB administrative view" and the "SNAP_GET_DB_V95 table function" on page 750 return snapshot information from the database (dbase) logical group.

SNAPDB administrative view

This administrative view allows you to retrieve snapshot information from the dbase logical group for the currently connected database.

Used in conjunction with the SNAPDB_MEMORY_POOL, SNAPDETAILLOG, SNAPHADR and SNAPSTORAGE_PATHS administrative views, the SNAPDB administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE on database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 206 on page 753 for a complete list of information that is returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDB administrative view
- CONTROL privilege on the SNAPDB administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DB_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON

- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve the status, platform, location, and connect time for all database partitions of the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDB ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME     DB_STATUS    SERVER_PLATFORM DB_LOCATION  ...
-------...- ------------ --------------- ------------ ...
TEST        ACTIVE       AIX64           LOCAL        ...
TEST        ACTIVE       AIX64           LOCAL        ...
TEST        ACTIVE       AIX64           LOCAL        ...

3 record(s) selected.
```

Output from this query (continued).

```
... DB_CONN_TIME               DBPARTITIONNUM
... -------------------------- --------------
... 2006-01-08-16.48.30.665477              0
... 2006-01-08-16.48.34.005328              1
... 2006-01-08-16.48.34.007937              2
```

This routine can be used by calling the following on the command line:

```
SELECT TOTAL_OLAP_FUNCS, OLAP_FUNC_OVERFLOWS, ACTIVE_OLAP_FUNCS
   FROM SYSIBMADM.SNAPDB

TOTAL_OLAP_FUNCS     OLAP_FUNC_OVERFLOWS  ACTIVE_OLAP_FUNCS
-------------------- -------------------- -----------------
                   7                    2                 1

  1 record(s) selected.
```

After running a workload, a user can use the following query:

```
SELECT STATS_CACHE_SIZE, STATS_FABRICATIONS, SYNC_RUNSTATS,
   ASYNC_RUNSTATS, STATS_FABRICATE_TIME, SYNC_RUNSTATS_TIME
   FROM SYSIBMADM.SNAPDB

STATS_CACHE_SIZE STATS_FABRICATIONS SYNC_RUNSTATS ASYNC_RUNSTATS ...
---------------- ------------------ ------------- -------------- ...
             128                  2             1              0 ...


... STATS_FABRICATE_TIME SYNC_RUNSTATS_TIME
... -------------------- ------------------
...                   10                100

1 record(s) selected.
```

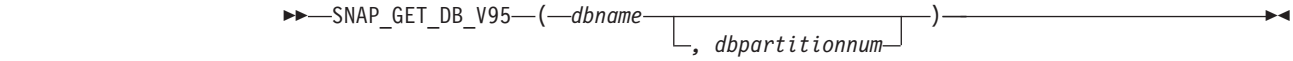
SNAP_GET_DB_V95 table function

The SNAP_GET_DB_V95 table function returns the same information as the SNAPDB administrative view.

Used in conjunction with the SNAP_GET_DB_MEMORY_POOL, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DB_V95 table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 206 on page 753 for a complete list of information that is returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DB_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DB_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Example 1: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions of the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME FROM TABLE(SNAP_GET_DB_V95('', -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME     DB_STATUS    SERVER_PLATFORM ...
-------...- ------------ -------------- ...
SAMPLE      ACTIVE       AIX64          ...

  1 record(s) selected.
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.09.22.013196
```

Example 2: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions for all active databases in the same instance that contains the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME
   FROM TABLE(SNAP_GET_DB_V95(CAST (NULL AS VARCHAR(128)), -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME      DB_STATUS    SERVER_PLATFORM ...
--------...- ------------ -------------- ...
TOOLSDB      ACTIVE       AIX64          ...
SAMPLE       ACTIVE       AIX64          ...
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.26.54.396335
... LOCAL        2005-07-24-22.09.22.013196
```

Example 3: This routine can be used by calling the following on the command line:

When connected to a database:

```
SELECT TOTAL_OLAP_FUNCS, OLAP_FUNC_OVERFLOWS, ACTIVE_OLAP_FUNCS
   FROM TABLE (SNAP_GET_DB_V95('', 0)) AS T

The output will look like:

TOTAL_OLAP_FUNCS  OLAP_FUNC_OVERFLOWS  ACTIVE_OLAP_FUNCS
----------------  -------------------- --------------------
               7                    2                    1

1 record(s) selected.
```

Example 4: After running a workload, a user can use the following query with the table function.

```
 SELECT STATS_CACHE_SIZE, STATS_FABRICATIONS, SYNC_RUNSTATS,
   ASYNC_RUNSTATS, STATS_FABRICATE_TIME, SYNC_RUNSTATS_TIME
   FROM TABLE (SNAP_GET_DB_V95('mytestdb', -1)) AS SNAPDB

STATS_CACHE_SIZE STATS_FABRICATIONS SYNC_RUNSTATS ASYNC_RUNSTATS ...
```

```
--------------- ----------------- ------------ ------------- ...
            200                 1            2             0 ...

Continued

...STATS_FABRICATE_TIME  SYNC_RUNSTATS_TIME
...----------------------  ------------------
...                     2                  32

  1 record(s) selected.
```

SNAPDB administrative view and SNAP_GET_DB_V95 table function metadata

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
DB_STATUS	VARCHAR(12)	db_status - Status of database. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ACTIVE • QUIESCE_PEND • QUIESCED • ROLLFWD • ACTIVE_STANDBY • STANDBY
CATALOG_PARTITION	SMALLINT	catalog_node - Catalog node number
CATALOG_PARTITION_NAME	VARCHAR(128)	catalog_node_name - Catalog node network name

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
SERVER_PLATFORM	VARCHAR(12)	server_platform - Server operating system. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS
DB_LOCATION	VARCHAR(12)	db_location - Database location. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • LOCAL • REMOTE
DB_CONN_TIME	TIMESTAMP	db_conn_time - Database activation timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
LAST_BACKUP	TIMESTAMP	last_backup - Last backup timestamp

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
CONNECTIONS_TOP	BIGINT	connections_top - Maximum number of concurrent connections
TOTAL_CONS	BIGINT	total_cons - Connects since database activation
TOTAL_SEC_CONS	BIGINT	total_sec_cons - Secondary connections
APPLS_CUR_CONS	BIGINT	appls_cur_cons - Applications connected currently
APPLS_IN_DB2	BIGINT	appls_in_db2 - Applications executing in the database currently
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
AGENTS_TOP	BIGINT	agents_top - Number of agents created
COORD_AGENTS_TOP	BIGINT	coord_agents_top - Maximum number of coordinating agents
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_LIST_IN_USE	BIGINT	lock_list_in_use - Total lock list memory in use
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
LOCKS_WAITING	BIGINT	locks_waiting - Current agents waiting on locks
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
NUM_INDOUBT_TRANS	BIGINT	num_indoubt_trans - Number of indoubt transactions
SORT_HEAP_ALLOCATED	BIGINT	sort_heap_allocated - Total sort heap allocated
SORT_SHRHEAP_ALLOCATED	BIGINT	sort_shrheap_allocated - Sort share heap currently allocated
SORT_SHRHEAP_TOP	BIGINT	sort_shrheap_top - Sort share heap high water mark
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
TOTAL_SORTS	BIGINT	total_sorts - Total sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
ACTIVE_SORTS	BIGINT	active_sorts - Active sorts
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
POOL_LSN_GAP_CLNS	BIGINT	pool_lsn_gap_clns - Buffer pool log space cleaners triggered
POOL_DRTY_PG_STEAL_CLNS	BIGINT	pool_drty_pg_steal_clns - Buffer pool victim page cleaners triggered
POOL_DRTY_PG_THRSH_CLNS	BIGINT	pool_drty_pg_thrsh_clns - Buffer pool threshold cleaners triggered
PREFETCH_WAIT_TIME	BIGINT	prefetch_wait_time - Time waited for prefetch
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
FILES_CLOSED	BIGINT	files_closed - Database files closed
ELAPSED_EXEC_TIME_S	BIGINT	elapsed_exec_time - Statement execution elapsed time (in seconds)*
ELAPSED_EXEC_TIME_MS	BIGINT	elapsed_exec_time - Statement execution elapsed time (fractional, in microseconds)*
COMMIT_SQL_STMTS	BIGINT	commit_sql_stmts - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	rollback_sql_stmts - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	dynamic_sql_stmts - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	static_sql_stmts - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	failed_sql_stmts - Failed statement operations
SELECT_SQL_STMTS	BIGINT	select_sql_stmts - Select SQL statements executed

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
UID_SQL_STMTS	BIGINT	uid_sql_stmts - UPDATE/INSERT/DELETE SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements
INT_AUTO_REBINDS	BIGINT	int_auto_rebinds - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
INT_COMMITS	BIGINT	int_commits - Internal commits
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	int_deadlock_rollbacks - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_SELECTED	BIGINT	rows_selected - Rows selected
ROWS_READ	BIGINT	rows_read - Rows read
BINDS_PRECOMPILES	BIGINT	binds_precompiles - Binds/precompiles attempted
TOTAL_LOG_AVAILABLE	BIGINT	total_log_available - Total log available
TOTAL_LOG_USED	BIGINT	total_log_used - Total log space used
SEC_LOG_USED_TOP	BIGINT	sec_log_used_top - Maximum secondary log space used
TOT_LOG_USED_TOP	BIGINT	tot_log_used_top - Maximum total log space used
SEC_LOGS_ALLOCATED	BIGINT	sec_logs_allocated - Secondary logs allocated currently
LOG_READS	BIGINT	log_reads - Number of log pages read
LOG_READ_TIME_S	BIGINT	log_read_time - Log read time (in seconds)†
LOG_READ_TIME_NS	BIGINT	log_read_time - Log read time (fractional, in nanoseconds)†
LOG_WRITES	BIGINT	log_writes - Number of log pages written
LOG_WRITE_TIME_S	BIGINT	log_write_time - Log write time (in seconds)†
LOG_WRITE_TIME_NS	BIGINT	log_write_time - Log write time (fractional, in nanoseconds)†
NUM_LOG_WRITE_IO	BIGINT	num_log_write_io - Number of log writes
NUM_LOG_READ_IO	BIGINT	num_log_read_io - Number of log reads
NUM_LOG_PART_PAGE_IO	BIGINT	num_log_part_page_io - Number of partial log page writes
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
NUM_LOG_DATA_FOUND_IN_BUFFER	BIGINT	num_log_data_found_in_buffer - Number of log data found in buffer
APPL_ID_OLDEST_XACT	BIGINT	appl_id_oldest_xact - Application with oldest transaction
LOG_TO_REDO_FOR_RECOVERY	BIGINT	log_to_redo_for_recovery - Amount of log to be redone for recovery
LOG_HELD_BY_DIRTY_PAGES	BIGINT	log_held_by_dirty_pages - Amount of log space accounted for by dirty pages
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
PKG_CACHE_NUM_OVERFLOWS	BIGINT	pkg_cache_num_overflows - Package cache overflows
PKG_CACHE_SIZE_TOP	BIGINT	pkg_cache_size_top - Package cache high water mark
APPL_SECTION_LOOKUPS	BIGINT	appl_section_lookups - Section lookups
APPL_SECTION_INSERTS	BIGINT	appl_section_inserts - Section inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	cat_cache_overflows - Catalog cache overflows
CAT_CACHE_SIZE_TOP	BIGINT	cat_cache_size_top - Catalog cache high water mark
PRIV_WORKSPACE_SIZE_TOP	BIGINT	priv_workspace_size_top - Maximum private workspace size
PRIV_WORKSPACE_NUM_OVERFLOWS	BIGINT	priv_workspace_num_overflows - Private workspace overflows
PRIV_WORKSPACE_SECTION_INSERTS	BIGINT	priv_workspace_section_inserts - Private workspace section inserts
PRIV_WORKSPACE_SECTION_LOOKUPS	BIGINT	priv_workspace_section_lookups - Private workspace section lookups
SHR_WORKSPACE_SIZE_TOP	BIGINT	shr_workspace_size_top - Maximum shared workspace size
SHR_WORKSPACE_NUM_OVERFLOWS	BIGINT	shr_workspace_num_overflows - Shared workspace overflows
SHR_WORKSPACE_SECTION_INSERTS	BIGINT	shr_workspace_section_inserts - Shared workspace section inserts

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
SHR_WORKSPACE_SECTION_LOOKUPS	BIGINT	shr_workspace_section_lookups - Shared workspace section lookups
TOTAL_HASH_JOINS	BIGINT	total_hash_joins - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	hash_join_overflows - Hash join overflows
HASH_JOIN_SMALL_OVERFLOWS	BIGINT	hash_join_small_overflows - Hash join small overflows
POST_SHRTHRESHOLD_HASH_JOINS	BIGINT	post_shrthreshold_hash_joins - Post threshold hash joins
ACTIVE_HASH_JOINS	BIGINT	active_hash_joins - Active hash joins
NUM_DB_STORAGE_PATHS	BIGINT	num_db_storage_paths - Number of automatic storage paths
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
SMALLEST_LOG_AVAIL_NODE	INTEGER	smallest_log_avail_node - Node with least available log space
TOTAL_OLAP_FUNCS	BIGINT	The total number of OLAP functions executed.
OLAP_FUNC_OVERFLOWS	BIGINT	The number of times that OLAP function data exceeded the available sort heap space.
ACTIVE_OLAP_FUNCS	BIGINT	The total number of OLAP functions that are currently running and consuming sort heap memory.
STATS_CACHE_SIZE	BIGINT	The size of the statistics cache in bytes
STATS_FABRICATIONS	BIGINT	Total number of statistics-collect activities for creating statistics by the system without table or index scan.
SYNC_RUNSTATS	BIGINT	Total number of synchronous statistics-collect activities during query compilation.
ASYNC_RUNSTATS	BIGINT	We will change the output for this column to total number of successful asynchronous statistics-collect activities.
STATS_FABRICATE_TIME	BIGINT	Total time spent on creating statistics by system without table or index scan during query compilation in milliseconds.
SYNC_RUNSTATS_TIME	BIGINT	Total time spent on synchronous statistics-collect activities in milliseconds.
NUM_THRESHOLD_VIOLATIONS	BIGINT	The number of threshold violations that have occurred at the database.

Table 206. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000. †To calculate the total elapsed time for this monitor element, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000,000 + *monitor-element-name*_MS) ÷ 1,000,000,000. For example, (LOG_READ_TIME_S × 1,000,000,000 + LOG_READ_TIME_MS) ÷ 1,000,000,000.		

SNAPDB_MEMORY_POOL administrative view and SNAP_GET_DB_MEMORY_POOL table function – Retrieve database level memory usage information

The SNAPDB_MEMORY_POOL administrative view and the SNAP_GET_DB_MEMORY_POOL table function return information about memory usage at the database level for UNIX platforms only.

SNAPDB_MEMORY_POOL administrative view

This administrative view allows you to retrieve database level memory usage information for the currently connected database.

Used with the SNAPDB, SNAPDETAILLOG, SNAPHADR and SNAPSTORAGE_PATHS administrative views, the SNAPDB_MEMORY_POOL administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 177 on page 624 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDB_MEMORY_POOL administrative view
- CONTROL privilege on the SNAPDB_MEMORY_POOL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DB_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size for the currently connected database, SAMPLE.

```
SELECT POOL_ID, POOL_CUR_SIZE FROM SYSIBMADM.SNAPDB_MEMORY_POOL
```

The following is an example of output from this query.

```
POOL_ID       POOL_CUR_SIZE
------------- --------------------
UTILITY                      32768
PACKAGE_CACHE               475136
CAT_CACHE                    65536
BP                         2097152
BP                         1081344
BP                          540672
BP                          278528
BP                          147456
BP                           81920
LOCK_MGR                    294912
DATABASE                   3833856
OTHER                            0

  12 record(s) selected.
```

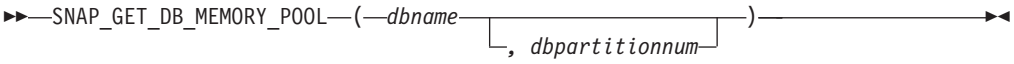
SNAP_GET_DB_MEMORY_POOL table function

The SNAP_GET_DB_MEMORY_POOL table function returns the same information as the SNAPDB_MEMORY_POOL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DB_V95, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DB_MEMORY_POOL table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 177 on page 624 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as

returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DB_MEMORY_POOL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DB_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of memory pools and their current size for all databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, POOL_ID, POOL_CUR_SIZE
   FROM TABLE(SNAPSHOT_GET_DB_MEMORY_POOL
   (CAST(NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  POOL_ID        POOL_CUR_SIZE
-------- -------------- --------------------
TESTDB   UTILITY                  65536
TESTDB   PACKAGE_CACHE           851968
TESTDB   CAT_CACHE                65536
TESTDB   BP                    35913728
TESTDB   BP                      589824
TESTDB   BP                      327680
TESTDB   BP                      196608
TESTDB   BP                      131072
TESTDB   SHARED_SORT              65536
TESTDB   LOCK_MGR              10092544
TESTDB   DATABASE               4980736
TESTDB   OTHER                   196608
SAMPLE   UTILITY                  65536
```

```
SAMPLE   PACKAGE_CACHE                655360
SAMPLE   CAT_CACHE                    131072
SAMPLE   BP                          4325376
SAMPLE   BP                           589824
SAMPLE   BP                           327680
SAMPLE   BP                           196608
SAMPLE   BP                           131072
SAMPLE   SHARED_SORT                       0
SAMPLE   LOCK_MGR                     655360
SAMPLE   DATABASE                    4653056
SAMPLE   OTHER                        196608

24 record(s) selected.
```

Information returned

Table 207. Information returned by the SNAPDB_MEMORY_POOL administrative view and the SNAP_GET_DB_MEMORY_POOL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
POOL_ID	VARCHAR(14)	pool_id - Memory pool identifier. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • APP_GROUP • APPL_CONTROL • APPLICATION • BP • CAT_CACHE • DATABASE • DFM • FCMBP • IMPORT_POOL • LOCK_MGR • MONITOR • OTHER • PACKAGE_CACHE • QUERY • SHARED_SORT • SORT • STATEMENT • STATISTICS • UTILITY
POOL_SECONDARY_ID	VARCHAR(32)	pool_secondary_id - Memory pool secondary identifier
POOL_CUR_SIZE	BIGINT	pool_cur_size - Current size of memory pool
POOL_WATERMARK	BIGINT	pool_watermark - Memory pool watermark

Table 207. Information returned by the SNAPDB_MEMORY_POOL administrative view and the SNAP_GET_DB_MEMORY_POOL table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_CONFIG_SIZE	BIGINT	pool_config_size - Configured size of memory pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPDBM administrative view and SNAP_GET_DBM_V95 table function - Retrieve the dbm logical grouping snapshot information

The SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function return the snapshot monitor DB2 database manager (dbm) logical grouping information.

SNAPDBM administrative view

Used with the SNAPDBM_MEMORY_POOL, SNAPFCM, SNAPFCM_PART and SNAPSWITCHES administrative views, the SNAPDBM administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 178 on page 627 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDBM administrative view
- CONTROL privilege on the SNAPDBM administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DBM_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve database manager status and connection information for all database partitions.

```
SELECT DB2_STATUS, DB2START_TIME, LAST_RESET, LOCAL_CONS, REM_CONS_IN,
   (AGENTS_CREATED_EMPTY_POOL/AGENTS_FROM_POOL) AS AGENT_USAGE,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPDBM ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB2_STATUS   DB2START_TIME             LAST_RESET     ...
------------ ------------------------- ----------...- ...
ACTIVE       2006-01-06-14.59.59.059879              - ...
ACTIVE       2006-01-06-14.59.59.097605              - ...
ACTIVE       2006-01-06-14.59.59.062798              - ...

  3 record(s) selected.                                 ...
```

Output from this query (continued).

```
... LOCAL_CONS     REM_CONS_IN     AGENT_USAGE     DBPARTITIONNUM
... ----------...- -----------...- -----------...- --------------
...              1               1               0              0
...              0               0               0              1
...              0               0               0              2
```

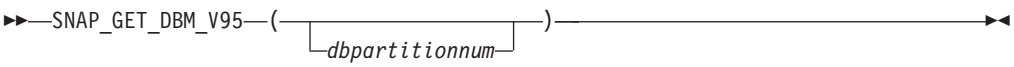
SNAP_GET_DBM_V95 table function

The SNAP_GET_DBM_V95 table function returns the same information as the SNAPDBM administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_MEMORY_POOL, SNAP_GET_FCM, SNAP_GET_FCM_PART and SNAP_GET_SWITCHES table functions, the SNAP_GET_DBM_V95 table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 178 on page 627 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DBM_V95 table function calls the snapshot from memory.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DBM_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the start time and current status of database partition number 2.

```
SELECT DB2START_TIME, DB2_STATUS FROM TABLE(SNAP_GET_DBM_V95(2)) AS T
```

The following is an example of output from this query.

```
DB2START_TIME              DB2_STATUS
-------------------------- ------------
2006-01-06-14.59.59.062798 ACTIVE
```

Information returned

Table 208. Information returned by the SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
SORT_HEAP_ALLOCATED	BIGINT	sort_heap_allocated - Total sort heap allocated
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
PIPED_SORTS_REQUESTED	BIGINT	piped_sorts_requested - Piped sorts requested
PIPED_SORTS_ACCEPTED	BIGINT	piped_sorts_accepted - Piped sorts accepted
REM_CONS_IN	BIGINT	rem_cons_in - Remote connections to database manager
REM_CONS_IN_EXEC	BIGINT	rem_cons_in_exec - Remote Connections Executing in the Database Manager monitor element
LOCAL_CONS	BIGINT	local_cons - Local connections
LOCAL_CONS_IN_EXEC	BIGINT	local_cons_in_exec - Local Connections Executing in the Database Manager monitor element
CON_LOCAL_DBASES	BIGINT	con_local_dbases - Local databases with current connects
AGENTS_REGISTERED	BIGINT	agents_registered - Agents registered
AGENTS_WAITING_ON_TOKEN	BIGINT	agents_waiting_on_token - Agents waiting for a token

Table 208. Information returned by the SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
DB2_STATUS	VARCHAR(12)	db2_status - Status of DB2 instance This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ACTIVE • QUIESCE_PEND • QUIESCED
AGENTS_REGISTERED_TOP	BIGINT	agents_registered_top - Maximum number of agents registered
AGENTS_WAITING_TOP	BIGINT	agents_waiting_top - Maximum number of agents waiting
COMM_PRIVATE_MEM	BIGINT	comm_private_mem - Committed private memory
IDLE_AGENTS	BIGINT	idle_agents - Number of idle agents
AGENTS_FROM_POOL	BIGINT	agents_from_pool - Agents assigned from pool
AGENTS_CREATED_EMPTY_POOL	BIGINT	agents_created_empty_pool - Agents created due to empty agent pool
COORD_AGENTS_TOP	BIGINT	coord_agents_top - Maximum number of coordinating agents
MAX_AGENT_OVERFLOWS	BIGINT	max_agent_overflows - Maximum agent overflows
AGENTS_STOLEN	BIGINT	agents_stolen - Stolen agents
GW_TOTAL_CONS	BIGINT	gw_total_cons - Total number of attempted connections for DB2 Connect
GW_CUR_CONS	BIGINT	gw_cur_cons - Current number of connections for DB2 Connect
GW_CONS_WAIT_HOST	BIGINT	gw_cons_wait_host - Number of connections waiting for the host to reply
GW_CONS_WAIT_CLIENT	BIGINT	gw_cons_wait_client - Number of connections waiting for the client to send request
POST_THRESHOLD_ HASH_JOINS	BIGINT	post_threshold_hash_joins - Hash join threshold
NUM_GW_CONN_SWITCHES	BIGINT	num_gw_conn_switches - Connection switches
DB2START_TIME	TIMESTAMP	db2start_time - Start database manager timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
NUM_NODES_IN_ DB2_INSTANCE	INTEGER	num_nodes_in_db2_instance - Number of nodes in database partition
PRODUCT_NAME	VARCHAR(32)	product_name - Product name
SERVICE_LEVEL	VARCHAR(18)	service_level - Service level
SORT_HEAP_TOP	BIGINT	sort_heap_top - Sort private heap high water mark

Table 208. Information returned by the SNAPDBM administrative view and the SNAP_GET_DBM_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
POST_THRESHOLD_OLAP_FUNCS	BIGINT	The number of OLAP functions which have requested a sort heap after the sort heap threshold has been exceeded. Sorts, hash joins, and OLAP functions are examples of operations which utilize a sort heap. Under normal conditions, the database manager will allocate sort heap using the value specified by the `sortheap` configuration parameter. If the amount of memory allocated to sort heaps exceeds the sort heap threshold (`sheapthres` configuration parameter), the database manager will allocate subsequent sort heaps using a value less than that specified by the `sortheap` configuration parameter. OLAP functions which start after the sort heap threshold has been reached may not receive an optimum amount of memory to execute.

SNAPDBM_MEMORY_POOL administrative view and SNAP_GET_DBM_MEMORY_POOL table function – Retrieve database manager level memory usage information

The SNAPDBM_MEMORY_POOL administrative view and the SNAP_GET_DBM_MEMORY_POOL table function return information about memory usage at the database manager.

SNAPDBM_MEMORY_POOL administrative view

Used with the SNAPDBM, SNAPFCM, SNAPFCM_PART and SNAPSWITCHES administrative views, the SNAPDBM_MEMORY_POOL administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 179 on page 631 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDBM_MEMORY_POOL administrative view
- CONTROL privilege on the SNAPDBM_MEMORY_POOL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DBM_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of the memory pools and their current size for the database manager of the connected database.

```
SELECT POOL_ID, POOL_CUR_SIZE FROM SNAPDBM_MEMORY_POOL
```

The following is an example of output from this query.

```
POOL_ID        POOL_CUR_SIZE
-------------- --------------------
MONITOR                   65536
OTHER                  29622272
FCMBP                  57606144
...
```

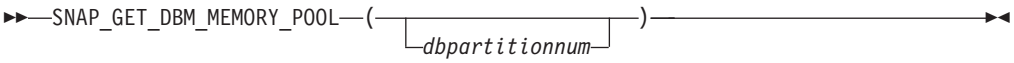
SNAP_GET_DBM_MEMORY_POOL table function

The SNAP_GET_DBM_MEMORY_POOL table function returns the same information as the SNAPDBM_MEMORY_POOL administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_V95, SNAP_GET_FCM, SNAP_GET_FCM_PART and SNAP_GET_SWITCHES table functions, the SNAP_GET_DBM_MEMORY_POOL table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 179 on page 631 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data

will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DBM_MEMORY_POOL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DBM_MEMORY_POOL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of the memory pools and their current size for all database partitions of the database manager of the connected database.

```
SELECT POOL_ID, POOL_CUR_SIZE, DBPARTITIONNUM
   FROM TABLE(SYSPROC.SNAP_GET_DBM_MEMORY_POOL())
   AS T ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
POOL_ID        POOL_CUR_SIZE        DBPARTITIONNUM
-------------- -------------------- --------------
MONITOR                   65536              0
OTHER                  29622272              0
FCMBP                  57606144              0
MONITOR                   65536              1
OTHER                  29425664              1
FCMBP                  57606144              1
MONITOR                   65536              2
OTHER                  29425664              2
FCMBP                  57606144              2
```

Information returned

Table 209. Information returned by the SNAPDBM_MEMORY_POOL administrative view and the SNAP_GET_DBM_MEMORY_POOL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.

Table 209. Information returned by the SNAPDBM_MEMORY_POOL administrative view and the SNAP_GET_DBM_MEMORY_POOL table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ID	VARCHAR(14)	pool_id - Memory pool identifier. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • APP_GROUP • APPL_CONTROL • APPLICATION • BP • CAT_CACHE • DATABASE • DFM • FCMBP • IMPORT_POOL • LOCK_MGR • MONITOR • OTHER • PACKAGE_CACHE • QUERY • SHARED_SORT • SORT • STATEMENT • STATISTICS • UTILITY
POOL_CUR_SIZE	BIGINT	pool_cur_size - Current size of memory pool
POOL_WATERMARK	BIGINT	pool_watermark - Memory pool watermark
POOL_CONFIG_SIZE	BIGINT	pool_config_size - Configured size of memory pool
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPDETAILLOG administrative view and SNAP_GET_DETAILLOG_V91 table function - Retrieve snapshot information from the detail_log logical data group

The SNAPDETAILLOG administrative view and the SNAP_GET_DETAILLOG_V91 table function return snapshot information from the detail_log logical data group.

SNAPDETAILLOG administrative view

This administrative view allows you to retrieve snapshot information from the detail_log logical data group for the currently connected database.

Used in conjunction with the SNAPDB, SNAPDB_MEMORY_POOL, SNAPHADR and SNAPSTORAGE_PATHS administrative views, the SNAPDETAILLOG administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE on database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 180 on page 635 for a complete list of information that is returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDETAILLOG administrative view
- CONTROL privilege on the SNAPDETAILLOG administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DETAILLOG_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve log information for all database partitions for the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DB_NAME, FIRST_ACTIVE_LOG,
   LAST_ACTIVE_LOG, CURRENT_ACTIVE_LOG, CURRENT_ARCHIVE_LOG,
   DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDETAILLOG ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME  FIRST_ACTIVE_LOG     LAST_ACTIVE_LOG      ...
-------- -------------------- -------------------- ...
TEST                        0                    8 ...
TEST                        0                    8 ...
TEST                        0                    8 ...

  3 record(s) selected.
```

Output from this query (continued).

```
... CURRENT_ACTIVE_LOG   CURRENT_ARCHIVE_LOG  DBPARTITIONNUM
... -------------------- -------------------- --------------
...                    0                    -              0
...                    0                    -              1
...                    0                    -              2
```

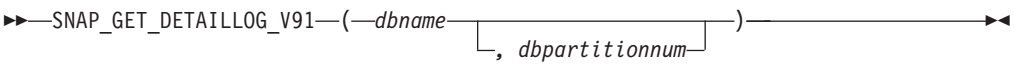
SNAP_GET_DETAILLOG_V91 table function

The SNAP_GET_DETAILLOG_V91 table function returns the same information as the SNAPDETAILLOG administrative view.

Used in conjunction with the SNAP_GET_DB_V95, SNAP_GET_DB_MEMORY_POOL, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DETAILLOG table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 180 on page 635 for a complete list of information that is returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DETAILLOG_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DETAILLOG_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve log information for database partition 1 for the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DB_NAME, FIRST_ACTIVE_LOG,
   LAST_ACTIVE_LOG, CURRENT_ACTIVE_LOG, CURRENT_ARCHIVE_LOG
   FROM TABLE(SNAP_GET_DETAILLOG_V91('', 1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  FIRST_ACTIVE_LOG     LAST_ACTIVE_LOG      ...
-------- -------------------- -------------------- ...
TEST                        0                    8 ...

  1 record(s) selected.
```

Output from this query (continued).

```
... CURRENT_ACTIVE_LOG   CURRENT_ARCHIVE_LOG
... -------------------- --------------------
...                    0                    -
```

SNAPDETAILLOG administrative view and SNAP_GET_DETAILLOG_V91 table function metadata

Table 210. Information returned by the SNAPDETAILLOG administrative view and SNAP_GET_DETAILLOG_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
FIRST_ACTIVE_LOG	BIGINT	first_active_log - First active log file number
LAST_ACTIVE_LOG	BIGINT	last_active_log - Last active log file number
CURRENT_ACTIVE_LOG	BIGINT	current_active_log - Current active log file number
CURRENT_ARCHIVE_LOG	BIGINT	current_archive_log - Current archive log file number
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPDYN_SQL administrative view and SNAP_GET_DYN_SQL_V95 table function - Retrieve dynsql logical group snapshot information

The "SNAPDYN_SQL administrative view" on page 636 and the "SNAP_GET_DYN_SQL_V95 table function" on page 637 return snapshot information from the dynsql logical data group.

SNAPDYN_SQL administrative view

This administrative view allows you to retrieve dynsql logical group snapshot information for the currently connected database.

This view returns information equivalent to the GET SNAPSHOT FOR DYNAMIC SQL ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 181 on page 639 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDYN_SQL administrative view
- CONTROL privilege on the SNAPDYN_SQL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DYN_SQL_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of dynamic SQL run on all database partitions of the currently connected database, ordered by the number of rows read.

```
SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT, 1, 60)
   AS STMT_TEXT, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDYN_SQL ORDER BY ROWS_READ
```

The following is an example of output from this query.

```
PREP_TIME_WORST      NUM_COMPILATIONS     ...
-------------------- -------------------- ...
                  98                    1 ...
                   9                    1 ...
                   0                    0 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                  40                    1 ...

  9 record(s) selected.
```

Output from this query (continued).

```
... STMT_TEXT                                                          ...
... ---------------------------------------------------------------- ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
... select * from dbuser.employee                                      ...
... SET CURRENT LOCALE LC_CTYPE = 'en_US'                              ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
... select * from dbuser.employee                                      ...
... insert into dbuser.employee values(1)                              ...
... select * from dbuser.employee                                      ...
... insert into dbuser.employee values(1)                              ...
```

Output from this query (continued).

```
... DBPARTITIONNUM
... --------------
...               0
...               0
...               0
...               2
...               1
...               2
...               2
...               1
...               0
```

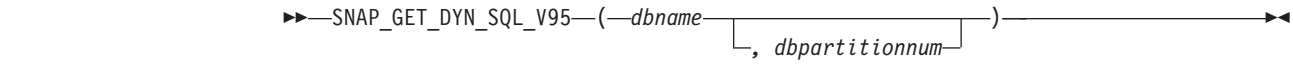
SNAP_GET_DYN_SQL_V95 table function

The SNAP_GET_DYN_SQL_V95 table function returns the same information as the SNAPDYN_SQL administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

This table function returns information equivalent to the GET SNAPSHOT FOR DYNAMIC SQL ON database-alias CLP command.

Refer to Table 181 on page 639 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input

option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DYN_SQL_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DYN_SQL_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of dynamic SQL run on the currently connected database partition of the currently connected database, ordered by the number of rows read.

```
SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT, 1, 60)
   AS STMT_TEXT FROM TABLE(SNAP_GET_DYN_SQL_V95('',-1)) as T
    ORDER BY ROWS_READ
```

The following is an example of output from this query.

```
PREP_TIME_WORST      ...
-------------------- ...
                   0 ...
                   3 ...
                     ...
                   4 ...
                     ...
                   4 ...
                     ...
                   4 ...
                     ...
                   3 ...
                     ...
                   4 ...
                     ...
```

Output from this query (continued).

```
... NUM_COMPILATIONS     STMT_TEXT
... -------------------- ----------------------------------------...-
...                    0 SET CURRENT LOCALE LC_CTYPE = 'en_US'
...                    1 select rows_read, rows_written,
...                          substr(stmt_text, 1, 40) as
...                    1 select * from table
...                          (snap_get_dyn_sqlv9('',-1)) as t
...                    1 select * from table
```

```
...                         (snap_getdetaillog9('',-1)) as t
...                     1 select * from table
...                         (snap_get_hadr('',-1)) as t
...                     1 select prep_time_worst, num_compilations,
...                         substr(stmt_text,
...                     1 select prep_time_worst, num_compilations,
...                         substr(stmt_text,
```

After running a workload, user can use the following query with the table function.

```
SELECT STATS_FABRICATE_TIME,SYNC_RUNSTATS_TIME
   FROM TABLE (SNAP_GET_DYN_SQL_V95('mytestdb', -1))
   AS SNAPDB

STATS_FABRICATE_TIME   SYNC_RUNSTATS_TIME
---------------------- ------------------
                     2                 12
                     1                 30
```

For the view based on this table function:

```
SELECT STATS_FABRICATE_TIME,SYNC_RUNSTATS_TIME
   FROM SYSIBMADM.SNAPDYN_SQL

STATS_FABRICATE_TIME   SYNC_RUNSTATS_TIME
---------------------- ------------------
                     5                 10
                     3                 20
  2 record(s) selected.
```

Information returned

Table 211. Information returned by the SNAPDYN_SQL administrative view and the SNAP_GET_DYN_SQL_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
NUM_EXECUTIONS	BIGINT	num_executions - Statement executions
NUM_COMPILATIONS	BIGINT	num_compilations - Statement compilations
PREP_TIME_WORST	BIGINT	prep_time_worst - Statement worst preparation time
PREP_TIME_BEST	BIGINT	prep_time_best - Statement best preparation time
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
STMT_SORTS	BIGINT	stmt_sorts - Statement sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads

Table 211. Information returned by the SNAPDYN_SQL administrative view and the SNAP_GET_DYN_SQL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
TOTAL_EXEC_TIME	BIGINT	total_exec_time - Elapsed statement execution time (in seconds)*
TOTAL_EXEC_TIME_MS	BIGINT	total_exec_time - Elapsed statement execution time (fractional, in microseconds)*
TOTAL_USR_CPU_TIME	BIGINT	total_usr_cpu_time - Total user CPU for a statement (in seconds)*
TOTAL_USR_CPU_TIME_MS	BIGINT	total_usr_cpu_time - Total user CPU for a statement (fractional, in microseconds)*
TOTAL_SYS_CPU_TIME	BIGINT	total_sys_cpu_time - Total system CPU for a statement (in seconds)*
TOTAL_SYS_CPU_TIME_MS	BIGINT	total_sys_cpu_time - Total system CPU for a statement (fractional, in microseconds)*
STMT_TEXT	CLOB(2 M)	stmt_text - SQL statement text
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
STATS_FABRICATE_TIME	BIGINT	The total time (in milliseconds) spent by system to create needed statistics without table or index scan during query compilation for a dynamic statement.
SYNC_RUNSTATS_TIME	BIGINT	The total time (in milliseconds) spent on synchronous statistics-collect activities during query compilation for a dynamic statement.

Table 211. Information returned by the SNAPDYN_SQL administrative view and the SNAP_GET_DYN_SQL_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPFCM administrative view and SNAP_GET_FCM table function – Retrieve the fcm logical data group snapshot information

The SNAPFCM administrative view and the SNAP_GET_FCM table function return information about the fast communication manager from a database manager snapshot, in particular, the fcm logical data group.

SNAPFCM administrative view

Used with the SNAPDBM, SNAPDBM_MEMORY_POOL, SNAPFCM_PART and SNAPSWITCHES administrative views, the SNAPFCM administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 182 on page 643 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPFCM administrative view
- CONTROL privilege on the SNAPFCM administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_FCM table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information about the fast communication manager's message buffers on all database partitions.

```
SELECT BUFF_FREE, BUFF_FREE_BOTTOM, DBPARTITIONNUM
    FROM SYSIBMADM.SNAPFCM ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
BUFF_FREE        BUFF_FREE_BOTTOM     DBPARTITIONNUM
---------...---- -------------------- --------------
            5120                 5100              0
            5120                 5100              1
            5120                 5100              2
```

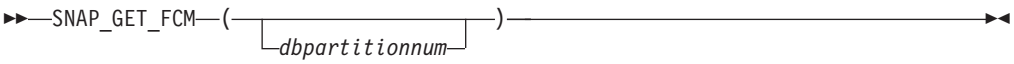
SNAP_GET_FCM table function

The SNAP_GET_FCM table function returns the same information as the SNAPFCM administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_V95, SNAP_GET_DBM_MEMORY_POOL, SNAP_GET_FCM_PART and SNAP_GET_SWITCHES table functions, the SNAP_GET_FCM table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 182 on page 643 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_FCM table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_FCM table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information about the fast communication manager's message buffers on database partition 1.

```
SELECT BUFF_FREE, BUFF_FREE_BOTTOM, DBPARTITIONNUM
   FROM TABLE(SYSPROC.SNAP_GET_FCM( 1 )) AS T
```

The following is an example of output from this query.

```
BUFF_FREE            BUFF_FREE_BOTTOM     DBPARTITIONNUM
-------------------- -------------------- --------------
                5120                 5100              1
```

Information returned

Table 212. Information returned by the SNAPFCM administrative view and the SNAP_GET_FCM table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BUFF_FREE	BIGINT	buff_free - FCM buffers currently free
BUFF_FREE_BOTTOM	BIGINT	buff_free_bottom - Minimum FCM Buffers Free
CH_FREE	BIGINT	ch_free - Channels Currently Free
CH_FREE_BOTTOM	BIGINT	ch_free_bottom - Minimum Channels Free
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPFCM_PART administrative view and SNAP_GET_FCM_PART table function – Retrieve the fcm_node logical data group snapshot information

The SNAPFCM_PART administrative view and the SNAP_GET_FCM_PART table function return information about the fast communication manager from a database manager snapshot, in particular, the fcm_node logical data group.

SNAPFCM_PART administrative view

Used with the SNAPDBM, SNAPDBM_MEMORY_POOL, SNAPFCM and SNAPSWITCHES administrative views, the SNAPFCM_PART administrative view provides the data equivalent to the GET SNAPSHOT FOR DBM command.

The schema is SYSIBMADM.

Refer to Table 183 on page 646 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPFCM_PART administrative view
- CONTROL privilege on the SNAPFCM_PART administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_FCM_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve buffers sent and received information for the fast communication manager.

```
SELECT CONNECTION_STATUS, TOTAL_BUFFERS_SENT, TOTAL_BUFFERS_RECEIVED
   FROM SYSIBMADM.SNAPFCM_PART WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
CONNECTION_STATUS    TOTAL_BUFFERS_SENT   TOTAL_BUFFERS_RCVD
-------------------- -------------------- --------------------
INACTIVE                               2                    1

  1 record(s) selected.
```

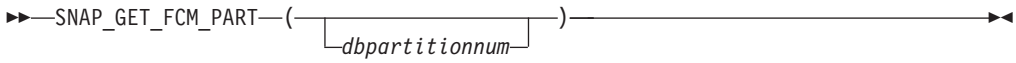
SNAP_GET_FCM_PART table function

The SNAP_GET_FCM_PART table function returns the same information as the SNAPFCM_PART administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DBM_V95, SNAP_GET_DBM_MEMORY_POOL, SNAP_GET_FCM and SNAP_GET_SWITCHES table functions, the SNAP_GET_FCM_PART table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 183 on page 646 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_FCM_PART table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_FCM_PART table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve buffers sent and received information for the fast communication manager for all database partitions.

```
SELECT FCM_DBPARTITIONNUM, TOTAL_BUFFERS_SENT, TOTAL_BUFFERS_RCVD,
   DBPARTITIONNUM FROM TABLE(SNAP_GET_FCM_PART()) AS T
   ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
FCM_DBPARTITIONNUM TOTAL_BUFFERS_SENT   TOTAL_BUFFERS_RCVD   DBPARTITIONNUM
------------------ -------------------- -------------------- --------------
                 0                  305                  305              0
                 1                 5647                 1664              0
                 2                 5661                 1688              0
                 0                   19                   19              1
                 1                  305                  301              1
                 2                 1688                 5661              1
```

```
0                1664              5647              2
1                  10                10              2
2                 301               305              2
```

Information returned

Table 213. Information returned by the SNAPFCM_PART administrative view and the SNAP_GET_FCM_PART table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
CONNECTION_STATUS	VARCHAR(10)	connection_status - Connection status. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • INACTIVE • ACTIVE • CONGESTED
TOTAL_BUFFERS_SENT	BIGINT	total_buffers_sent - Total FCM buffers sent
TOTAL_BUFFERS_RCVD	BIGINT	total_buffers_rcvd - Total FCM buffers received
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
FCM_DBPARTITIONNUM	SMALLINT	The database partition number to which data was sent or from which data was received (as per the TOTAL_BUFFERS_SENT and TOTAL_BUFFERS_RCVD columns).

SNAPHADR administrative view and SNAP_GET_HADR table function – Retrieve hadr logical data group snapshot information

The SNAPHADR administrative view and the SNAP_GET_HADR table function return information about high availability disaster recovery from a database snapshot, in particular, the hadr logical data group.

SNAPHADR administrative view

This administrative view allows you to retrieve hadr logical data group snapshot information for the currently connected database. The data is only returned by this view if the database is a primary or standby high availability disaster recovery (HADR) database.

Used with the SNAPDB, SNAPDB_MEMORY_POOL, SNAPDETAILLOG and SNAPSTORAGE_PATHS administrative views, the SNAPHADR administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 184 on page 649 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPHADR administrative view
- CONTROL privilege on the SNAPHADR administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_HADR table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the configuration and status information for HADR on the primary HADR database.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DBNAME, HADR_ROLE, HADR_STATE,
   HADR_SYNCMODE, HADR_CONNECT_STATUS
   FROM SYSIBMADM.SNAPHADR
```

The following is an example of output from this query.

```
DBNAME   HADR_ROLE HADR_STATE     HADR_SYNCMODE HADR_CONNECT_STATUS
-------- --------- -------------- ------------- -------------------
SAMPLE   PRIMARY   PEER           SYNC          CONNECTED

  1 record(s) selected.
```

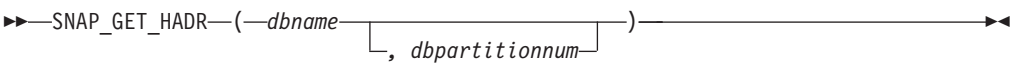
SNAP_GET_HADR table function

The SNAP_GET_HADR table function returns the same information as the SNAPHADR administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DB_V95, SNAP_GET_DB_MEMORY_POOL, SNAP_GET_DETAILLOG_V91 and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_HADR table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 184 on page 649 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to *NULL* and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_HADR table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_HADR table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the configuration and status information for HADR for all databases.

```
SELECT SUBSTR(DB_NAME, 1, 8) AS DBNAME, HADR_ROLE, HADR_STATE,
   HADR_SYNCMODE, HADR_CONNECT_STATUS
   FROM TABLE (SNAP_GET_HADR (CAST (NULL as VARCHAR(128)), 0)) as T
```

The following is an example of output from this query.

```
DBNAME   HADR_ROLE HADR_STATE     HADR_SYNCMODE HADR_CONNECT_STATUS
-------- --------- -------------- ------------- ------------------
SAMPLE   PRIMARY   PEER           SYNC          CONNECTED
TESTDB   PRIMARY   DISCONNECTED   NEARSYNC      DISCONNECTED

2 record(s) selected.
```

Information returned

Table 214. Information returned by the SNAPHADR administrative view and the SNAP_GET_HADR table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
HADR_ROLE	VARCHAR(10)	hadr_role - HADR role. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • PRIMARY • STANDARD • STANDBY
HADR_STATE	VARCHAR(14)	hadr_state - HADR state. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • DISCONNECTED • LOCAL_CATCHUP • PEER • REM_CATCH_PEN • REM_CATCHUP
HADR_SYNCMODE	VARCHAR(10)	hadr_syncmode - HADR synchronization mode. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • ASYNC • NEARSYNC • SYNC
HADR_CONNECT_STATUS	VARCHAR(12)	hadr_connect_status - HADR connection status. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • CONGESTED • CONNECTED • DISCONNECTED
HADR_CONNECT_TIME	TIMESTAMP	hadr_connect_time - HADR connection time
HADR_HEARTBEAT	INTEGER	hadr_heartbeat - HADR heartbeat
HADR_LOCAL_HOST	VARCHAR(255)	hadr_local_host - HADR local host

Table 214. Information returned by the SNAPHADR administrative view and the SNAP_GET_HADR table function (continued)

Column name	Data type	Description or corresponding monitor element
HADR_LOCAL_SERVICE	VARCHAR(40)	hadr_local_service - HADR local service
HADR_REMOTE_HOST	VARCHAR(255)	hadr_remote_host - HADR remote host
HADR_REMOTE_SERVICE	VARCHAR(40)	hadr_remote_service - HADR remote service
HADR_REMOTE_INSTANCE	VARCHAR(128)	hadr_remote_instance - HADR remote instance
HADR_TIMEOUT	BIGINT	hadr_timeout - HADR timeout
HADR_PRIMARY_LOG_FILE	VARCHAR(255)	hadr_primary_log_file - HADR primary log file
HADR_PRIMARY_LOG_PAGE	BIGINT	hadr_primary_log_page - HADR primary log page
HADR_PRIMARY_LOG_LSN	BIGINT	hadr_primary_log_lsn - HADR primary log LSN
HADR_STANDBY_LOG_FILE	VARCHAR(255)	hadr_standby_log_file - HADR standby log file
HADR_STANDBY_LOG_PAGE	BIGINT	hadr_standby_log_page - HADR standby log page
HADR_STANDBY_LOG_LSN	BIGINT	hadr_standby_log_lsn - HADR standby log LSN
HADR_LOG_GAP	BIGINT	hadr_log_gap - HADR log gap
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPLOCK administrative view and SNAP_GET_LOCK table function – Retrieve lock logical data group snapshot information

Note: This administrative view and table function have been deprecated and replaced by the "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415, "MON_GET_LOCKS - list all locks in the currently connected database" on page 442, and "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381.

The SNAPLOCK administrative view and the SNAP_GET_LOCK table function return snapshot information about locks, in particular, the lock logical data group.

SNAPLOCK administrative view

This administrative view allows you to retrieve lock logical data group snapshot information for the currently connected database.

Used with the SNAPLOCKWAIT administrative view, the SNAPLOCK administrative view provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 185 on page 653 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPLOCK administrative view
- CONTROL privilege on the SNAPLOCK administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_LOCK table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock information for the database partition 0 of the currently connected database.

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
   FROM SYSIBMADM.SNAPLOCK WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
AGENT_ID             LOCK_OBJECT_TYPE LOCK_MODE LOCK_STATUS
-------------------- ---------------- --------- -----------
                   7 TABLE            IX        GRNT

  1 record(s) selected.
```

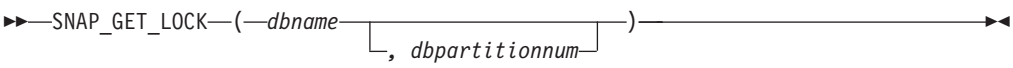
SNAP_GET_LOCK table function

The SNAP_GET_LOCK table function returns the same information as the SNAPLOCK administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_LOCKWAIT table function, the SNAP_GET_LOCK table function provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

Refer to Table 185 on page 653 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify a null value or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_LOCK table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_LOCK table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock information for the current database partition of the currently connected database.

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
   FROM TABLE(SNAP_GET_LOCK('',-1)) as T
```

The following is an example of output from this query.

```
AGENT_ID       LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS
--------...--- ------------------ ---------- -----------
           680 INTERNALV_LOCK     S          GRNT
           680 INTERNALP_LOCK     S          GRNT

2 record(s) selected.
```

Information returned

Table 215. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
TAB_FILE_ID	BIGINT	table_file_id - Table file identification
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK

Table 215. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_STATUS	VARCHAR(10)	lock_status - Lock status. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • CONV • GRNT
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
TABNAME	VARCHAR(128)	table_name - Table name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
LOCK_ATTRIBUTES	VARCHAR(128)	lock_attributes - Lock attributes. This interface returns a text identifier based on the defines in `sqlmon.h`. If there are no locks, the text identifier is NONE, otherwise, it is any combination of the following separated by a '+' sign: • ALLOW_NEW • DELETE_IN_BLOCK • ESCALATED • INSERT • NEW_REQUEST • RR • RR_IN_BLOCK • UPDATE_DELETE • WAIT_FOR_AVAIL
LOCK_COUNT	BIGINT	lock_count - Lock count

Table 215. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_CURRENT_MODE	VARCHAR(10)	lock_current_mode - Original lock mode before conversion. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_HOLD_COUNT	BIGINT	lock_hold_count - Lock hold count
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
LOCK_RELEASE_FLAGS	BIGINT	lock_release_flags - Lock release flags
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element is NULL.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function – Retrieve lockwait logical data group snapshot information

Note: This administrative view and table function have been deprecated and replaced by the "MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503 and the "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415, "MON_GET_LOCKS - list all locks in the currently connected database" on page 442, and "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381.

The SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function return snapshot information about lock waits, in particular, the lockwait logical data group.

SNAPLOCKWAIT administrative view

This administrative view allows you to retrieve lockwait logical data group snapshot information for the currently connected database.

Used with the SNAPLOCK administrative view, the SNAPLOCKWAIT administrative view provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 186 on page 658 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPLOCKWAIT administrative view
- CONTROL privilege on the SNAPLOCKWAIT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_LOCKWAIT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock wait information on database partition 0 for the currently connected database.

```
SELECT AGENT_ID, LOCK_MODE, LOCK_OBJECT_TYPE, AGENT_ID_HOLDING_LK,
   LOCK_MODE_REQUESTED FROM SYSIBMADM.SNAPLOCKWAIT
   WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
AGENT_ID     LOCK_MODE LOCK_OBJECT_TYPE ...
--------...- --------- ---------------- ...
           7 IX        TABLE            ...

  1 record(s) selected.
```

Output from this query (continued).

```
... AGENT_ID_HOLDING_LK  LOCK_MODE_REQUESTED
... -------------------- ------------------
...                   12 IS
```

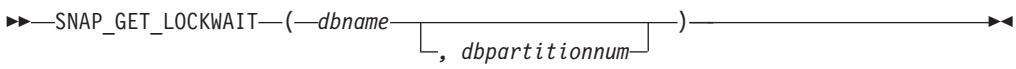
SNAP_GET_LOCKWAIT table function

The SNAP_GET_LOCKWAIT table function returns the same information as the SNAPLOCKWAIT administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_LOCK table function, the SNAP_GET_LOCKWAIT table function provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

Refer to Table 186 on page 658 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify a null value or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_LOCKWAIT table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_LOCKWAIT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock wait information on current database partition for the currently connected database.

```
SELECT AGENT_ID, LOCK_MODE, LOCK_OBJECT_TYPE, AGENT_ID_HOLDING_LK,
   LOCK_MODE_REQUESTED FROM TABLE(SNAP_GET_LOCKWAIT('',-1)) AS T
```

The following is an example of output from this query.

```
AGENT_ID      LOCK_MODE  LOCK_OBJECT_TYPE   ...
--------...-- ---------- ------------------ ...
          12 X          ROW_LOCK           ...

1 record(s) selected.
```

Output from this query (continued).

```
... AGENT_ID_HOLDING_LK  LOCK_MODE_REQUESTED
... -------------------- ------------------
...                    7 X
```

Usage note

To see lock wait information, you must first turn on the default LOCK monitor switch in the database manager configuration. To have the change take effect immediately explicitly attach to the instance using CLP and then issue the CLP command:

```
UPDATE DATABASE MANAGER CONFIGURATION CLP USING DFT_MON_LOCK ON
```

The default setting can also be turned on through the ADMIN_CMD stored procedure. For example:

```
CALL SYSPROC.ADMIN_CMD('update dbm cfg using DFT_MON_LOCK ON')
```

If the ADMIN_CMD stored procedure is used or if the clp command is used without having previously attached to the instance, the instance must be recycled before the change takes effect.

Information returned

Table 216. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
SUBSECTION_NUMBER	BIGINT	ss_number - Subsection number

Table 216. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
AGENT_ID_HOLDING_LK	BIGINT	agent_id_holding_lock - Agent ID holding lock

Table 216. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_WAIT_START_TIME	TIMESTAMP	lock_wait_start_time - Lock wait start timestamp
LOCK_MODE_REQUESTED	VARCHAR(10)	lock_mode_requested - Lock mode requested. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
TABNAME	VARCHAR(128)	table_name - Table name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
APPL_ID_HOLDING_LK	VARCHAR(128)	appl_id_holding_lk - Application ID holding lock
LOCK_ATTRIBUTES	VARCHAR(128)	lock_attributes - Lock attributes. This interface returns a text identifier based on the defines in `sqlmon.h`. If there are no locks, the text identifier is NONE, otherwise, it is any combination of the following separated by a '+' sign: • ALLOW_NEW • DELETE_IN_BLOCK • ESCALATED • INSERT • NEW_REQUEST • RR • RR_IN_BLOCK • UPDATE_DELETE • WAIT_FOR_AVAIL

Table 216. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_CURRENT_MODE	VARCHAR(10)	lock_current_mode - Original lock mode before conversion. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
LOCK_RELEASE_FLAGS	BIGINT	lock_release_flags - Lock release flags.
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element is NULL.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPSTMT administrative view and SNAP_GET_STMT table function – Retrieve statement snapshot information

The SNAPSTMT administrative view and the SNAP_GET_STMT table function return information about SQL or XQuery statements from an application snapshot.

SNAPSTMT administrative view

This administrative view allows you to retrieve statement snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPAPPL_INFO and SNAPSUBSECTION administrative views, the SNAPSTMT administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS on database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 187 on page 664 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:
- SELECT privilege on the SNAPSTMT administrative view
- CONTROL privilege on the SNAPSTMT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the SNAP_GET_STMT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve rows read, written and operation performed for statements executed on the currently connected single-partition database.

```
SELECT SUBSTR(STMT_TEXT,1,30) AS STMT_TEXT, ROWS_READ, ROWS_WRITTEN,
   STMT_OPERATION FROM SYSIBMADM.SNAPSTMT
```

The following is an example of output from this query.

```
STMT_TEXT     ROWS_READ     ROWS_WRITTEN     STMT_OPERATION
---------...- ---------...- ------------...- --------------------
-                         0                0 FETCH
-                         0                0 STATIC_COMMIT

  2 record(s) selected.
```

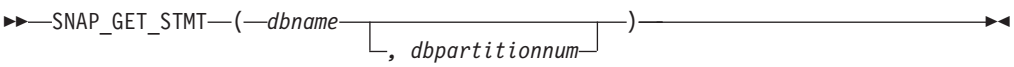
SNAP_GET_STMT table function

The SNAP_GET_STMT table function returns the same information as the SNAPSTMT administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95 and SNAP_GET_SUBSECTION table functions, the SNAP_GET_STMT table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 187 on page 664 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_STMT table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_STMT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve rows read, written and operation performed for statements executed on current database partition of currently connected database.

```
SELECT SUBSTR(STMT_TEXT,1,30) AS STMT_TEXT, ROWS_READ,
   ROWS_WRITTEN, STMT_OPERATION FROM TABLE(SNAP_GET_STMT('',-1)) AS T
```

The following is an example of output from this query.

```
STMT_TEXT                      ROWS_READ     ...
------------------------------ ---------...- ...
update t set a=3                          0 ...
SELECT SUBSTR(STMT_TEXT,1,30)             0 ...
-                                         0 ...
-                                         0 ...
```

```
update t set a=2                          9 ...
                                            ...
5 record(s) selected.                       ...
```

Output from this query (continued).

```
... ROWS_WRITTEN     STMT_OPERATION
... ------------...- --------------------
...                0 EXECUTE_IMMEDIATE
...                0 FETCH
...                0 NONE
...                0 NONE
...                1 EXECUTE_IMMEDIATE
...
```

Information returned

Table 217. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
NUM_AGENTS	BIGINT	num_agents - Number of agents working on a statement
AGENTS_TOP	BIGINT	agents_top - Number of agents created
STMT_TYPE	VARCHAR(20)	stmt_type - Statement type. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • DYNAMIC • NON_STMT • STATIC • STMT_TYPE_UNKNOWN

Table 217. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function (continued)

Column name	Data type	Description or corresponding monitor element
STMT_OPERATION	VARCHAR(20)	stmt_operation/operation - Statement operation. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • CALL • CLOSE • COMPILE • DESCRIBE • EXECUTE • EXECUTE_IMMEDIATE • FETCH • FREE_LOCATOR • GETAA • GETNEXTCHUNK • GETTA • NONE • OPEN • PREP_COMMIT • PREP_EXEC • PREP_OPEN • PREPARE • REBIND • REDIST • REORG • RUNSTATS • SELECT • SET • STATIC_COMMIT • STATIC_ROLLBACK
SECTION_NUMBER	BIGINT	section_number - Section number
QUERY_COST_ESTIMATE	BIGINT	query_cost_estimate - Query cost estimate
QUERY_CARD_ESTIMATE	BIGINT	query_card_estimate - Query number of rows estimate
DEGREE_PARALLELISM	BIGINT	degree_parallelism - Degree of parallelism
STMT_SORTS	BIGINT	stmt_sorts - Statement sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated

Table 217. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function (continued)

Column name	Data type	Description or corresponding monitor element
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
FETCH_COUNT	BIGINT	fetch_count - Number of successful fetches
STMT_START	TIMESTAMP	stmt_start - Statement operation start timestamp
STMT_STOP	TIMESTAMP	stmt_stop - Statement operation stop timestamp
STMT_USR_CPU_TIME_S	BIGINT	stmt_usr_cpu_time - User CPU time used by statement (in seconds)*
STMT_USR_CPU_TIME_MS	BIGINT	stmt_usr_cpu_time - User CPU time used by statement (fractional, in microseconds)*
STMT_SYS_CPU_TIME_S	BIGINT	stmt_sys_cpu_time - System CPU time used by statement (in seconds)*
STMT_SYS_CPU_TIME_MS	BIGINT	stmt_sys_cpu_time - System CPU time used by statement (fractional, in microseconds)*
STMT_ELAPSED_TIME_S	BIGINT	stmt_elapsed_time - Most recent statement elapsed time (in seconds)*
STMT_ELAPSED_TIME_MS	BIGINT	stmt_elapsed_time - Most recent statement elapsed time (fractional, in microseconds)*
BLOCKING_CURSOR	SMALLINT	blocking_cursor - Blocking cursor
STMT_NODE_NUMBER	SMALLINT	stmt_node_number - Statement node
CURSOR_NAME	VARCHAR(128)	cursor_name - Cursor name
CREATOR	VARCHAR(128)	creator - Application creator
PACKAGE_NAME	VARCHAR(128)	package_name - Package name
STMT_TEXT	CLOB(16 M)	stmt_text - SQL statement text
CONSISTENCY_TOKEN	VARCHAR(128)	consistency_token - Package consistency token
PACKAGE_VERSION_ID	VARCHAR(128)	package_version_id - Package version
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads

Table 217. Information returned by the SNAPSTMT administrative view and the SNAP_GET_STMT table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads monitor element
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads monitor element
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPSTORAGE_PATHS administrative view and SNAP_GET_STORAGE_PATHS table function - Retrieve automatic storage path information

Note: This table function has been deprecated and replaced by "SNAPSTORAGE_PATHS administrative view and SNAP_GET_STORAGE_PATHS_V97 table function - Retrieve automatic storage path information" on page 667.

The SNAPSTORAGE_PATHS administrative view and the SNAP_GET_STORAGE_PATHS table function return a list of automatic storage paths for the database including file system information for each storage path, specifically, from the db_storage_group logical data group.

SNAPSTORAGE_PATHS administrative view

This administrative view allows you to retrieve automatic storage path information for the currently connected database.

Used with the SNAPDB, SNAPDETAILLOG, SNAPHADR and SNAPDB_MEMORY_POOL administrative views, the SNAPSTORAGE_PATHS administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 218 on page 810 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPSTORAGE_PATHS administrative view
- CONTROL privilege on the SNAPSTORAGE_PATHS administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_STORAGE_PATHS table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the storage path for the currently connected single-partition database.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, SUBSTR(DB_STORAGE_PATH,1,8)
   AS DB_STORAGE_PATH FROM SYSIBMADM.SNAPSTORAGE_PATHS
```

The following is an example of output from this query.

```
DB_NAME  DB_STORAGE_PATH
-------- ---------------
STOPATH  d:

  1 record(s) selected.
```

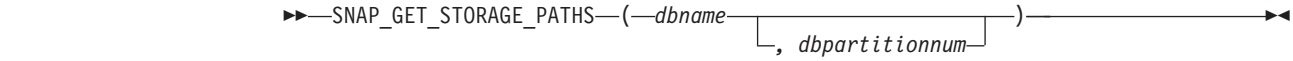
SNAP_GET_STORAGE_PATHS table function

The SNAP_GET_STORAGE_PATHS table function returns the same information as the SNAPSTORAGE_PATHS administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_DB_V95, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_DB_MEMORY_POOL table functions, the SNAP_GET_STORAGE_PATHS table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 218 on page 810 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`,`-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_STORAGE_PATHS table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_STORAGE_PATHS table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve the storage path information for all active databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, DB_STORAGE_PATH
   FROM TABLE(SNAP_GET_STORAGE_PATHS(CAST (NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  DB_STORAGE_PATH
-------- -------------------...
STOPATH  /home/jessicae/sdb
MYDB     /home/jessicae/mdb

  2 record(s) selected
```

Information returned

The BUFFERPOOL monitor switch must be turned on in order for the file system information to be returned.

Table 218. Information returned by the SNAPSTORAGE_PATHS administrative view and the SNAP_GET_STORAGE_PATHS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
DB_STORAGE_PATH	VARCHAR(256)	db_storage_path - Automatic storage path
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
FS_ID	VARCHAR(22)	fs_id - Unique file system identification number
FS_TOTAL_SIZE	BIGINT	fs_total_size - Total size of a file system
FS_USED_SIZE	BIGINT	fs_used_size - Amount of space used on a file system
STO_PATH_FREE_SIZE	BIGINT	sto_path_free_sz - Automatic storage path free space

SNAPSUBSECTION administrative view and SNAP_GET_SUBSECTION table function – Retrieve subsection logical monitor group snapshot information

The SNAPSUBSECTION administrative view and the SNAP_GET_SUBSECTION table function return information about application subsections, namely the subsection logical monitor grouping.

SNAPSUBSECTION administrative view

This administrative view allows you to retrieve subsection logical monitor group snapshot information for the currently connected database.

Used with the SNAPAGENT, SNAPAGENT_MEMORY_POOL, SNAPAPPL, SNAPAPPL_INFO and SNAPSTMT administrative views, the SNAPSUBSECTION administrative view provides information equivalent to the GET SNAPSHOT FOR APPLICATIONS on database-alias CLP command, but retrieves data from all database partitions.

The schema is SYSIBMADM.

Refer to Table 189 on page 673 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPSUBSECTION administrative view
- CONTROL privilege on the SNAPSUBSECTION administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_SUBSECTION table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Get status for subsections executing on all database partitions.

```
SELECT DB_NAME, STMT_TEXT, SS_STATUS, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPSUBSECTION
   ORDER BY DB_NAME, SS_STATUS, DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME     STMT_TEXT                 SS_STATUS     DBPARTITIONNUM
-------...- ----------------------...- ---------...- --------------
SAMPLE      select * from EMPLOYEE    EXEC                       0
SAMPLE      select * from EMPLOYEE    EXEC                       1
```

SNAP_GET_SUBSECTION table function

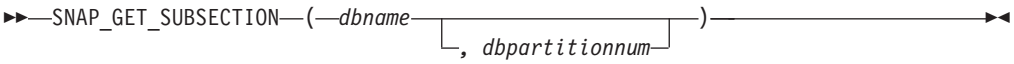
The SNAP_GET_SUBSECTION table function returns the same information as the SNAPSUBSECTION administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Refer to Table 189 on page 673 for a complete list of information that can be returned.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_V95, SNAP_GET_APPL_INFO_V95 and SNAP_GET_STMT

table functions, the SNAP_GET_SUBSECTION table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_SUBSECTION table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_SUBSECTION table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Get status for subsections executing on all database partitions.

```
SELECT DB_NAME, STMT_TEXT, SS_STATUS, DBPARTITIONNUM
   FROM TABLE(SYSPROC.SNAP_GET_SUBSECTION( '', 0 )) as T
   ORDER BY DB_NAME, SS_STATUS, DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME     STMT_TEXT                 SS_STATUS     DBPARTITIONNUM
-------...- ----------------------...- ----------...- --------------
SAMPLE      select * from EMPLOYEE    EXEC                       0
SAMPLE      select * from EMPLOYEE    EXEC                       1
```

Information returned

Table 219. Information returned by the SNAPSUBSECTION administrative view and the SNAP_GET_SUBSECTION table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
STMT_TEXT	CLOB(16 M)	stmt_text - SQL statement text
SS_EXEC_TIME	BIGINT	ss_exec_time - Subsection execution elapsed time
TQ_TOT_SEND_SPILLS	BIGINT	tq_tot_send_spills - Total number of table queue buffers overflowed
TQ_CUR_SEND_SPILLS	BIGINT	tq_cur_send_spills - Current number of table queue buffers overflowed
TQ_MAX_SEND_SPILLS	BIGINT	tq_max_send_spills - Maximum number of table queue buffers overflows
TQ_ROWS_READ	BIGINT	tq_rows_read - Number of rows read from table queues
TQ_ROWS_WRITTEN	BIGINT	tq_rows_written - Number of rows written to table queues
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
SS_USR_CPU_TIME_S	BIGINT	ss_usr_cpu_time - User CPU time used by subsection (in seconds)*
SS_USR_CPU_TIME_MS	BIGINT	ss_usr_cpu_time - User CPU time used by subsection (fractional, in microseconds)*
SS_SYS_CPU_TIME_S	BIGINT	ss_sys_cpu_time - System CPU time used by subsection (in seconds)*
SS_SYS_CPU_TIME_MS	BIGINT	ss_sys_cpu_time - System CPU time used by subsection (fractional, in microseconds)*
SS_NUMBER	INTEGER	ss_number - Subsection number

Table 219. Information returned by the SNAPSUBSECTION administrative view and the SNAP_GET_SUBSECTION table function (continued)

Column name	Data type	Description or corresponding monitor element
SS_STATUS	VARCHAR(20)	ss_status - Subsection status. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • EXEC • TQ_WAIT_TO_RCV • TQ_WAIT_TO_SEND • COMPLETED
SS_NODE_NUMBER	SMALLINT	ss_node_number - Subsection node number
TQ_NODE_WAITED_FOR	SMALLINT	tq_node_waited_for - Waited for node on a table queue
TQ_WAIT_FOR_ANY	INTEGER	tq_wait_for_any - Waiting for any node to send on a table queue
TQ_ID_WAITING_ON	INTEGER	tq_id_waiting_on - Waited on node on a table queue
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPSWITCHES administrative view and SNAP_GET_SWITCHES table function – Retrieve database snapshot switch state information

The SNAPSWITCHES administrative view and the SNAP_GET_SWITCHES table function return information about the database snapshot switch state.

SNAPSWITCHES administrative view

This view provides the data equivalent to the GET DBM MONITOR SWITCHES CLP command.

The schema is SYSIBMADM.

Refer to Table 190 on page 676 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPSWITCHES administrative view
- CONTROL privilege on the SNAPSWITCHES administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_SWITCHES table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve DBM monitor switches state information for all database partitions.

```
SELECT UOW_SW_STATE, STATEMENT_SW_STATE, TABLE_SW_STATE, BUFFPOOL_SW_STATE,
   LOCK_SW_STATE, SORT_SW_STATE, TIMESTAMP_SW_STATE,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPSWITCHES
```

The following is an example of output from this query.

```
UOW_SW_STATE STATEMENT_SW_STATE TABLE_SW_STATE BUFFPOOL_SW_STATE ...
------------ ----------------- -------------- ----------------- ...
           0                 0              0                 0 ...
           0                 0              0                 0 ...
           0                 0              0                 0 ...
                                                                 ...
  3 record selected.
```

Output from this query (continued).

```
... LOCK_SW_STATE SORT_SW_STATE TIMESTAMP_SW_STATE DBPARTITIONNUM
... ------------- ------------- ------------------ --------------
...             1             0                  1              0
...             1             0                  1              1
...             1             0                  1              2
```

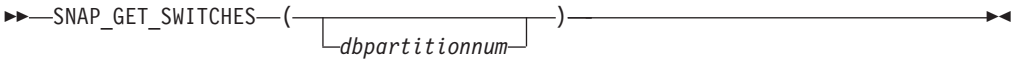
SNAP_GET_SWITCHES table function

The SNAP_GET_SWITCHES table function returns the same information as the SNAPSWITCHES administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

This table function provides the data equivalent to the GET DBM MONITOR SWITCHES CLP command.

Refer to Table 190 on page 676 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_SWITCHES table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_SWITCHES table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve DBM monitor switches state information for the current database partition.

```
SELECT UOW_SW_STATE, STATEMENT_SW_STATE, TABLE_SW_STATE,
   BUFFPOOL_SW_STATE,LOCK_SW_STATE, SORT_SW_STATE, TIMESTAMP_SW_STATE
   FROM TABLE(SNAP_GET_SWITCHES(-1)) AS T
```

The following is an example of output from this query.

```
UOW_SW_STATE STATEMENT_SW_STATE TABLE_SW_STATE...
------------ ----------------- -------------...
           1                 1             1...
                                            ...
1 record(s) selected.                       ...
```

Output from this query (continued).

```
... BUFFPOOL_SW_STATE LOCK_SW_STATE SORT_SW_STATE TIMESTAMP_SW_STATE
... ---------------- ------------- ------------- ------------------
...                1             1             0                  1
```

Information returned

Table 220. Information returned by the SNAPSWITCHES administrative view and the SNAP_GET_SWITCHES table function

Column name	Data type	Description
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
UOW_SW_STATE	SMALLINT	State of the unit of work monitor recording switch (0 or 1).
UOW_SW_TIME	TIMESTAMP	If the unit of work monitor recording switch is on, the date and time that this switch was turned on.
STATEMENT_SW_STATE	SMALLINT	State of the SQL statement monitor recording switch (0 or 1).
STATEMENT_SW_TIME	TIMESTAMP	If the SQL statement monitor recording switch is on, the date and time that this switch was turned on.
TABLE_SW_STATE	SMALLINT	State of the table activity monitor recording switch (0 or 1).
TABLE_SW_TIME	TIMESTAMP	If the table activity monitor recording switch is on, the date and time that this switch was turned on.
BUFFPOOL_SW_STATE	SMALLINT	State of the buffer pool activity monitor recording switch (0 or 1).
BUFFPOOL_SW_TIME	TIMESTAMP	If the buffer pool activity monitor recording switch is on, the date and time that this switch was turned on.
LOCK_SW_STATE	SMALLINT	State of the lock monitor recording switch (0 or 1).
LOCK_SW_TIME	TIMESTAMP	If the lock monitor recording switch is on, the date and time that this switch was turned on.
SORT_SW_STATE	SMALLINT	State of the sorting monitor recording switch (0 or 1).
SORT_SW_TIME	TIMESTAMP	If the sorting monitor recording switch is on, the date and time that this switch was turned on.
TIMESTAMP_SW_STATE	SMALLINT	State of the timestamp monitor recording switch (0 or 1)
TIMESTAMP_SW_TIME	TIMESTAMP	If the timestamp monitor recording switch is on, the date and time that this switch was turned on.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTAB administrative view and SNAP_GET_TAB_V91 table function - Retrieve table logical data group snapshot information

The SNAPTAB administrative view and the SNAP_GET_TAB_V91 table function return snapshot information from the table logical data group.

SNAPTAB administrative view

This administrative view allows you to retrieve table logical data group snapshot information for the currently connected database.

Used in conjunction with the SNAPTAB_REORG administrative view, the SNAPTAB administrative view returns equivalent information to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 191 on page 680 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTAB administrative view
- CONTROL privilege on the SNAPTAB administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TAB_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the schema and name for all active tables.

```
SELECT SUBSTR(TABSCHEMA,1,8), SUBSTR(TABNAME,1,15) AS TABNAME, TAB_TYPE,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPTAB
```

The following is an example of output from this query.

```
TABSCHEMA TABNAME         TAB_TYPE     DBPARTITIONNUM
--------- --------------- ------------ --------------
SYSTOOLS  HMON_ATM_INFO   USER_TABLE               0

  1 record selected.
```

SNAP_GET_TAB_V91 table function

The SNAP_GET_TAB_V91 table function returns the same information as the SNAPTAB administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_TAB_REORG table function, the SNAP_GET_TAB_V91 table function returns equivalent information to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

Refer to Table 191 on page 680 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify `NULL` or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TAB_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TAB_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON

- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of active tables as an aggregate view for the currently connected database.

```
SELECT SUBSTR(TABSCHEMA,1,8) AS TABSCHEMA, SUBSTR(TABNAME,1,15) AS TABNAME,
   TAB_TYPE, DBPARTITIONNUM FROM TABLE(SNAP_GET_TAB('',-2)) AS T
```

The following is an example of output from this query.

```
TABSCHEMA TABNAME         TAB_TYPE      DBPARTITIONNUM
--------- --------------- ------------- --------------
SYSTOOLS  HMON_ATM_INFO   USER_TABLE                 -
JESSICAE  EMPLOYEE        USER_TABLE                 -
```

Information returned

Table 221. Information returned by the SNAPTAB administrative view and the SNAP_GET_TAB_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
TAB_FILE_ID	BIGINT	table_file_id - Table file identification
TAB_TYPE	VARCHAR(14)	table_type - Table type. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • USER_TABLE • DROPPED_TABLE • TEMP_TABLE • CATALOG_TABLE • REORG_TABLE
DATA_OBJECT_PAGES	BIGINT	data_object_pages - Data object pages
INDEX_OBJECT_PAGES	BIGINT	index_object_pages - Index object pages
LOB_OBJECT_PAGES	BIGINT	lob_object_pages - LOB object pages
LONG_OBJECT_PAGES	BIGINT	long_object_pages - Long object pages
XDA_OBJECT_PAGES	BIGINT	xda_object_pages - XDA Object Pages
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written

Table 221. Information returned by the SNAPTAB administrative view and the SNAP_GET_TAB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
OVERFLOW_ACCESSES	BIGINT	overflow_accesses - Accesses to overflowed records
PAGE_REORGS	BIGINT	page_reorgs - Page reorganizations
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
TBSP_ID	BIGINT	tablespace_id - Table space identification
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element will be NULL.

SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function - Retrieve table reorganization snapshot information

The SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function return table reorganization information. If no tables have been reorganized, 0 rows are returned. When a data partitioned table is reorganized, one record for each data partition is returned. If only a specific data partition of a data partitioned table is reorganized, only a record the for the partition is returned.

SNAPTAB_REORG administrative view

This administrative view allows you to retrieve table reorganization snapshot information for the currently connected database.

Used with the SNAPTAB administrative view, the SNAPTAB_REORG administrative view provides the data equivalent to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 192 on page 684 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTAB_REORG administrative view
- CONTROL privilege on the SNAPTAB_REORG administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TAB_REORG table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Select details on reorganization operations for all database partitions on the currently connected database.

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
   AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
   REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TAB_NAME     TAB_SCHEMA     REORG_PHASE      ...
--------...- ----------...- ---------------- ...
EMPLOYEE     DBUSER         REPLACE          ...
EMPLOYEE     DBUSER         REPLACE          ...
EMPLOYEE     DBUSER         REPLACE          ...
                                             ...
3 record(s) selected.
```

Output from this query (continued).

```
... REORG_TYPE           REORG_STATUS REORG_COMPLETION DBPARTITIONNUM
... -------------------- ------------ ---------------- --------------
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       0
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       1
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       2
```

Select all information about a reorganization operation to reclaim extents from a multidimensional clustering (MDC) table from the SNAPTAB_REORG administrative view..

```
db2 -v "select * from sysibmadm.snaptab_reorg"

TABNAME  REORG_PHASE       REORG_MAX_PHASE   REORG_TYPE
-------- ----------------- ----------------- -------------------------
T1       RELEASE           3                 RECLAIM_EXTENTS+ALLOW_WRITE

REORG_STATUS REORG_COMPLETION REORG_START                REORG_END
------------ ---------------- -------------------------- --------------------------
COMPLETED    SUCCESS          2008-09-24-14.35.30.734741 2008-09-24-14.35.31.460674
```

SNAP_GET_TAB_REORG table function

The SNAP_GET_TAB_REORG table function returns the same information as the SNAPTAB_REORG administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TAB table function, the SNAP_GET_TAB_REORG table function provides the data equivalent to the GET SNAPSHOT FOR TABLES ON database-alias CLP command.

Refer to Table 192 on page 684 for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TAB_REORG table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TAB_REORG table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Select details on reorganization operations for database partition 1 on the currently connected database.

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
   AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
   REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
   FROM TABLE( SNAP_GET_TAB_REORG('', 1)) AS T
```

The following is an example of output from this query.

```
TAB_NAME     TAB_SCHEMA     REORG_PHASE     REORG_TYPE           ...
--------...- ----------...- ------------...- ------------------- ...
EMPLOYEE     DBUSER         REPLACE         RECLAIM+OFFLINE+ALLO ...
                                                                 ...
  1 record(s) selected.                                          ...
```

Output from this query (continued).

```
... REORG_STATUS REORG_COMPLETION DBPARTITIONNUM
... ------------ ---------------- --------------
... COMPLETED    SUCCESS                     1
...
```

Select all information about a reorganization operation to reclaim extents from a multidimensional clustering (MDC) table using the SNAP_GET_TAB_REORG table function.

```
db2 -v "select * from table(snap_get_tab_reorg(''))"

TABNAME  REORG_PHASE       REORG_MAX_PHASE   REORG_TYPE
-------- ----------------- ----------------- --------------------------
T1       RELEASE           3                 RECLAIM_EXTENTS+ALLOW_WRITE

REORG_STATUS REORG_COMPLETION REORG_START                REORG_END
------------ ---------------- -------------------------- --------------------------
COMPLETED    SUCCESS          2008-09-24-14.35.30.734741 2008-09-24-14.35.31.460674
```

Information returned

Table 222. Information returned by the SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABNAME	VARCHAR (128)	table_name - Table name
TABSCHEMA	VARCHAR (128)	table_schema - Table schema name
PAGE_REORGS	BIGINT	page_reorgs - Page reorganizations
REORG_PHASE	VARCHAR (16)	reorg_phase - Table reorganize phase. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • BUILD • DICT_SAMPLE • INDEX_RECREATE • REPLACE • SORT • SCAN • DRAIN • RELEASE or SORT+DICT_SAMPLE.
REORG_MAX_PHASE	INTEGER	reorg_max_phase - Maximum table reorganize phase
REORG_CURRENT_ COUNTER	BIGINT	reorg_current_counter - Table reorganize progress

Table 222. Information returned by the SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function (continued)

Column name	Data type	Description or corresponding monitor element
REORG_MAX_COUNTER	BIGINT	reorg_max_counter - Total amount of table reorganization

REORG_TYPE	VARCHAR (128)	reorg_type - Table reorganize attributes. This interface returns a text identifier using a combination of the following identifiers separated by '+':
		Either:
		• RECLAIM
		• RECLUSTER
		• RECLAIM_EXTS
		and either:
		• +OFFLINE
		• +ONLINE
		If access mode is specified, it is one of:
		• +ALLOW_NONE
		• +ALLOW_READ
		• +ALLOW_WRITE
		If offline and RECLUSTER option, one of:
		• +INDEXSCAN
		• +TABLESCAN
		If offline, one of:
		• +LONGLOB
		• +DATAONLY
		If offline, and option is specified, any of:
		• +CHOOSE_TEMP
		• +KEEPDICTIONARY
		• +RESETDICTIONARY
		If online, and option is specified:
		• +NOTRUNCATE
		Example 1: If a REORG TABLE TEST.EMPLOYEE was run, the following would be displayed:
		`RECLAIM+OFFLINE+ALLOW_READ+DATAONLY +KEEPDICTIONARY`
		Example 2: If a REORG TABLE TEST.EMPLOYEE INDEX EMPIDX INDEXSCAN was run, then the following would be displayed:
		`RECLUSTER+OFFLINE+ALLOW_READ+INDEXSCAN +DATAONLY+KEEPDICTIONARY`

Table 222. Information returned by the SNAPTAB_REORG administrative view and the SNAP_GET_TAB_REORG table function (continued)

Column name	Data type	Description or corresponding monitor element
REORG_STATUS	VARCHAR (10)	reorg_status - Table reorganize status. This interface returns a text identifier based on defines in `sqlmon.h` and is one of: • COMPLETED • PAUSED • STARTED • STOPPED • TRUNCATE
REORG_COMPLETION	VARCHAR (10)	reorg_completion - Table reorganization completion flag. This interface returns a text identifier, based on defines in `sqlmon.h` and is one of: • FAIL • SUCCESS
REORG_START	TIMESTAMP	reorg_start - Table reorganize start time
REORG_END	TIMESTAMP	reorg_end - Table reorganize end time
REORG_PHASE_START	TIMESTAMP	reorg_phase_start - Table reorganize phase start time
REORG_INDEX_ID	BIGINT	reorg_index_id - Index used to reorganize the table
REORG_TBSPC_ID	BIGINT	reorg_tbspc_id - Table space where table is reorganized
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element will be NULL.
REORG_ ROWSCOMPRESSED	BIGINT	reorg_rows_compressed - Rows compressed
REORG_ROWSREJECTED	BIGINT	reorg_rows_rejected_for_compression - Rows rejected for compression
REORG_LONG_TBSPC_ID	BIGINT	reorg_long_tbspc_id - Table space where long objects are reorganized

SNAPTBSP administrative view and SNAP_GET_TBSP_V91 table function - Retrieve table space logical data group snapshot information

The SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function return snapshot information from the table space logical data group.

SNAPTBSP administrative view

This administrative view allows you to retrieve table space logical data group snapshot information for the currently connected database.

Used in conjunction with the SNAPTBSP_PART, SNAPTBSP_QUIESCER, SNAPTBSP_RANGE, SNAPCONTAINER administrative views, the SNAPTBSP administrative view returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 193 on page 689 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTBSP administrative view
- CONTROL privilege on the SNAPTBSP administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TBSP_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of table spaces on the catalog database partition for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME,1,30) AS TBSP_NAME, TBSP_ID, TBSP_TYPE,
   TBSP_CONTENT_TYPE FROM SYSIBMADM.SNAPTBSP WHERE DBPARTITIONNUM = 1
```

The following is an example of output from this query.

```
TBSP_NAME  TBSP_ID     TBSP_TYPE  TBSP_CONTENT_TYPE
---------- -------...- ---------- ----------------
TEMPSPACE1           1 SMS        SYSTEMP
USERSPACE1           2 DMS        LONG

2 record(s) selected.
```

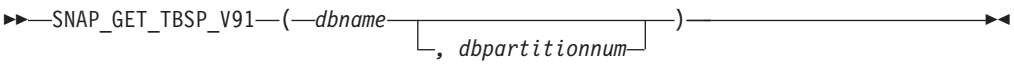
SNAP_GET_TBSP_V91 table function

The SNAP_GET_TBSP_V91 table function returns the same information as the SNAPTBSP administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_QUIESCER, SNAP_GET_TBSP_RANGE, SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_V91 table function returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 193 on page 689 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify `NULL` or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TBSP_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of table spaces for all database partitions for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME,1,10) AS TBSP_NAME, TBSP_ID, TBSP_TYPE,
   TBSP_CONTENT_TYPE, DBPARTITIONNUM FROM TABLE(SNAP_GET_TBSP_V91('')) AS T
```

The following is an example of output from this query.

```
TBSP_NAME  TBSP_ID     TBSP_TYPE  TBSP_CONTENT_TYPE DBPARTITIONNUM
---------- -------...- ---------- ----------------- -------------
TEMPSPACE1           1 SMS        SYSTEMP                         1
USERSPACE1           2 DMS        LONG                            1
SYSCATSPAC           0 DMS        ANY                             0
TEMPSPACE1           1 SMS        SYSTEMP                         0
USERSPACE1           2 DMS        LONG                            0
SYSTOOLSPA           3 DMS        LONG                            0
TEMPSPACE1           1 SMS        SYSTEMP                         2
USERSPACE1           2 DMS        LONG                            2

  8 record(s) selected.
```

Information returned

Table 223. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
TBSP_TYPE	VARCHAR(10)	tablespace_type - Table space type. This interface returns a text identifier based on defines in `sqlutil.h`, and is one of: • DMS • SMS
TBSP_CONTENT_TYPE	VARCHAR(10)	tablespace_content_type - Table space contents type. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ANY • LARGE • SYSTEMP • USRTEMP
TBSP_PAGE_SIZE	BIGINT	tablespace_page_size - Table space page size
TBSP_EXTENT_SIZE	BIGINT	tablespace_extent_size - Table space extent size
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size

Table 223. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_CUR_POOL_ID	BIGINT	tablespace_cur_pool_id - Buffer pool currently being used
TBSP_NEXT_POOL_ID	BIGINT	tablespace_next_pool_id - Buffer pool that will be used at next startup
FS_CACHING	SMALLINT	fs_caching - File system caching
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads

Table 223. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
FILES_CLOSED	BIGINT	files_closed - Database files closed
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages

Table 223. Information returned by the SNAPTBSP administrative view and the SNAP_GET_TBSP_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_REBALANCER_MODE	VARCHAR(10)	tablespace_rebalancer_mode - Rebalancer mode. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • NO_REBAL • FWD_REBAL • REV_REBAL
TBSP_USING_AUTO_STORAGE	SMALLINT	tablespace_using_auto_storage - Table space enabled for automatic storage
TBSP_AUTO_RESIZE_ENABLED	SMALLINT	tablespace_auto_resize_enabled - Table space automatic resizing enabled
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V91 table function - Retrieve tablespace_nodeinfo logical data group snapshot information

Note: This table function has been deprecated and replaced by "SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V97 table function - Retrieve tablespace_nodeinfo logical data group snapshot information" on page 692.

The SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V91 table function return snapshot information from the tablespace_nodeinfo logical data group.

SNAPTBSP_PART administrative view

This administrative view allows you to retrieve tablespace_nodeinfo logical data group snapshot information for the currently connected database.

Used in conjunction with the SNAPTBSP, SNAPTBSP_QUIESCER, SNAPTBSP_RANGE, SNAPCONTAINER administrative views, the SNAPTBSP_PART administrative view returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 224 on page 835 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTBSP_PART administrative view
- CONTROL privilege on the SNAPTBSP_PART administrative view

- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the SNAP_GET_TBSP_PART_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of table spaces and their state for all database partitions of the currently connected database.

```
SELECT SUBSTR(TBSP_NAME,1,30) AS TBSP_NAME, TBSP_ID,
   SUBSTR(TBSP_STATE,1,30) AS TBSP_STATE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTBSP_PART
```

The following is an example of output from this query.

```
TBSP_NAME       TBSP_ID     TBSP_STATE     DBPARTITIONNUM
-----------...- -------...- ----------...- --------------
SYSCATSPACE               0 NORMAL                       0
TEMPSPACE1                1 NORMAL                       0
USERSPACE1                2 NORMAL                       0
TEMPSPACE1                1 NORMAL                       1
USERSPACE1                2 NORMAL                       1

   5 record(s) selected.
```

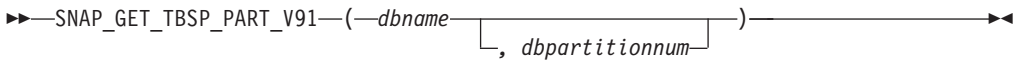
SNAP_GET_TBSP_PART_V91 table function

The SNAP_GET_TBSP_PART_V91 table function returns the same information as the SNAPTBSP_PART administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_TBSP_V91, SNAP_GET_TBSP_QUIESCER, SNAP_GET_TBSP_RANGE, SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_PART_V91 table function returns information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 224 on page 835 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_PART_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TBSP_PART_V91 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of table spaces and their state for the connected database partition of the connected database.

```
SELECT SUBSTR(TBSP_NAME,1,30) AS TBSP_NAME, TBSP_ID,
   SUBSTR(TBSP_STATE,1,30) AS TBSP_STATE
   FROM TABLE(SNAP_GET_TBSP_PART_V91(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
TBSP_NAME                      TBSP_ID              TBSP_STATE
------------------------------ -------------------- ------------...-
SYSCATSPACE                                       0 NORMAL
TEMPSPACE1                                        1 NORMAL
USERSPACE1                                        2 NORMAL
SYSTOOLSPACE                                      3 NORMAL
SYSTOOLSTMPSPACE                                  4 NORMAL

  5 record(s) selected.
```

Information returned

Table 224. Information returned by the SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR (128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
TBSP_STATE	VARCHAR (256)	tablespace_state - Table space state. This interface returns a text identifier based on defines in `sqlutil.h` and is combination of the following separated by a '+' sign: • BACKUP_IN_PROGRESS • BACKUP_PENDING • DELETE_PENDING • DISABLE_PENDING • DROP_PENDING • LOAD_IN_PROGRESS • LOAD_PENDING • NORMAL • OFFLINE • PSTAT_CREATION • PSTAT_DELETION • QUIESCED_EXCLUSIVE • QUIESCED_SHARE • QUIESCED_UPDATE • REBAL_IN_PROGRESS • REORG_IN_PROGRESS • RESTORE_IN_PROGRESS • RESTORE_PENDING • ROLLFORWARD_IN_PROGRESS • ROLLFORWARD_PENDING • STORDEF_ALLOWED • STORDEF_CHANGED • STORDEF_FINAL_VERSION • STORDEF_PENDING • SUSPEND_WRITE
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size
TBSP_NUM_QUIESCERS	BIGINT	tablespace_num_quiescers - Number of quiescers
TBSP_STATE_CHANGE_OBJECT_ID	BIGINT	tablespace_state_change_object_id - State change object identification
TBSP_STATE_CHANGE_TBSP_ID	BIGINT	tablespace_state_change_ts_id - State change table space identification

Table 224. Information returned by the SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_MIN_RECOVERY_TIME	TIMESTAMP	tablespace_min_recovery_time - Minimum recovery time for rollforward
TBSP_TOTAL_PAGES	BIGINT	tablespace_total_pages - Total pages in table space
TBSP_USABLE_PAGES	BIGINT	tablespace_usable_pages - Usable pages in table space
TBSP_USED_PAGES	BIGINT	tablespace_used_pages - Used pages in table space
TBSP_FREE_PAGES	BIGINT	tablespace_free_pages - Free pages in table space
TBSP_PENDING_FREE_PAGES	BIGINT	tablespace_pending_free_pages - Pending free pages in table space
TBSP_PAGE_TOP	BIGINT	tablespace_page_top - Table space high water mark
REBALANCER_MODE	VARCHAR (10)	tablespace_rebalancer_mode - Rebalancer mode. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • FWD_REBAL • NO_REBAL • REV_REBAL
REBALANCER_EXTENTS_REMAINING	BIGINT	tablespace_rebalancer_extents_remaining - Total number of extents to be processed by the rebalancer
REBALANCER_EXTENTS_PROCESSED	BIGINT	tablespace_rebalancer_extents_processed - Number of extents the rebalancer has processed
REBALANCER_PRIORITY	BIGINT	tablespace_rebalancer_priority - Current rebalancer priority
REBALANCER_START_TIME	TIMESTAMP	tablespace_rebalancer_start_time - Rebalancer start time
REBALANCER_RESTART_TIME	TIMESTAMP	tablespace_rebalancer_restart_time - Rebalancer restart time
REBALANCER_LAST_EXTENT_MOVED	BIGINT	tablespace_rebalancer_last_extent_moved - Last extent moved by the rebalancer
TBSP_NUM_RANGES	BIGINT	tablespace_num_ranges - Number of ranges in the table space map
TBSP_NUM_CONTAINERS	BIGINT	tablespace_num_containers - Number of containers in table space
TBSP_INITIAL_SIZE	BIGINT	tablespace_initial_size - Initial table space size

Table 224. Information returned by the SNAPTBSP_PART administrative view and the SNAP_GET_TBSP_PART_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_CURRENT_SIZE	BIGINT	tablespace_current_size - Current table space size
TBSP_MAX_SIZE	BIGINT	tablespace_max_size - Maximum table space size
TBSP_INCREASE_SIZE	BIGINT	tablespace_increase_size - Increase size in bytes
TBSP_INCREASE_SIZE_ PERCENT	SMALLINT	tablespace_increase_size_percent - Increase size by percent
TBSP_LAST_RESIZE_TIME	TIMESTAMP	tablespace_last_resize_time - Time of last successful resize
TBSP_LAST_RESIZE_ FAILED	SMALLINT	tablespace_last_resize_failed - Last resize attempt failed
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTBSP_QUIESCER administrative view and SNAP_GET_TBSP_QUIESCER table function - Retrieve quiescer table space snapshot information

The SNAPTBSP_QUIESCER administrative view and the SNAP_GET_TBSP_QUIESCER table function return information about quiescers from a table space snapshot.

SNAPTBSP_QUIESCER administrative view

This administrative view allows you to retrieve quiescer table space snapshot information for the currently connected database.

Used with the SNAPTBSP, SNAPTBSP_PART, SNAPTBSP_RANGE, SNAPCONTAINER administrative views, the SNAPTBSP_QUIESCER administrative view provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 195 on page 701 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTBSP_QUIESCER administrative view
- CONTROL privilege on the SNAPTBSP_QUIESCER administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TBSP_QUIESCER table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information on quiesced table spaces for all database partitions for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME, 1, 10) AS TBSP_NAME, QUIESCER_TS_ID,
   QUIESCER_OBJ_ID, QUIESCER_AUTH_ID, QUIESCER_AGENT_ID,
   QUIESCER_STATE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTBSP_QUIESCER ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TBSP_NAME  QUIESCER_TS_ID     QUIESCER_OBJ_ID     QUIESCER_AUTH_ID     ..
---------- --------------...- ---------------...- ----------------...- ..
USERSPACE1                 2                   5 SWALKTY              ..
USERSPACE1                 2                   5 SWALKTY              ..

  2 record(s) selected.
```

Output from this query (continued).

```
... QUIESCER_AGENT_ID    QUIESCER_STATE DBPARTITIONNUM
... -------------------- -------------- --------------
...                    0 EXCLUSIVE                  0
...                65983 EXCLUSIVE                  1
```

Example: Determine the range partitioned table names

If the table is range-partitioned and kept in quiesced state, the different values for table space ID and table ID are represented than in SYSCAT.TABLES. These IDs will appear as the unsigned short representation. In order to find the quiesced table name, you need to find the signed short representation first by calculating the table space ID that is subtracting 65536 (the maximum value) from QEUIESCER_TS_ID and then use this table space ID to locate the quiesced tables. (The actual table space ID can be found in SYSCAT.DATAPARTITIONS for each range partition in the table).

```
SELECT SUBSTR(TBSP_NAME, 1, 10) AS TBSP_NAME,
  CASE WHEN QUIESCER_TS_ID = 65530
    THEN QUIESCER_TS_ID - 65536
    ELSE QUIESCER_TS_ID END as tbspaceid,
  CASE WHEN QUIESCER_TS_ID = 65530
    THEN QUIESCER_OBJ_ID - 65536
    ELSE QUIESCER_OBJ_ID END as tableid
  FROM SYSIBMADM.SNAPTBSP_QUIESCER
  ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TBSP_NAME     TBSPACEID    TABLEID
------------- ------------ --------
TABDATA       -6           -32768
DATAMART      -6           -32765
SMALL         5            17

  3 record(s) selected.
```

Use the given TBSPACEID and TABLEID provided from above query to find the table schema and name from SYSCAT.TABLES.

```
SELECT CHAR(tabschema, 10)tabschema, CHAR(tabname,15)tabname
  FROM SYSCAT.TABLES
  WHERE tbspaceid = -6 AND tableid in (-32768,-32765)
```

The following is an example of output from this query.

```
TABSCHEMA    TABNAME
------------ --------
TPCD         ORDERS_RP
TPCD         ORDERS_DMART

  2 record(s) selected.
```

```
SELECT CHAR(tabschema, 10)tabschema, CHAR(tabname,15)tabname
  FROM SYSCAT.TABLES
  WHERE tbspaceid = 5 AND tableid = 17
```

The following is an example of output from this query.

```
TABSCHEMA    TABNAME
------------ --------
TPCD         NATION

  1 record(s) selected.
```

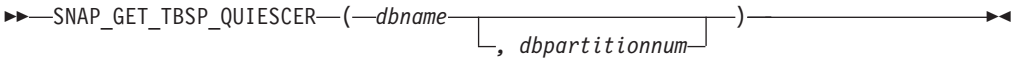
SNAP_GET_TBSP_QUIESCER table function

The SNAP_GET_TBSP_QUIESCER table function returns the same information as the SNAPTBSP_QUIESCER administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TBSP_V91, SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_RANGE, SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_QUIESCER table function provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 195 on page 701 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database

name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_QUIESCER table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TBSP_QUIESCER table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve information on quiesced table spaces for database partition 1 for the currently connected database.

```
SELECT SUBSTR(TBSP_NAME, 1, 10) AS TBSP_NAME, QUIESCER_TS_ID,
   QUIESCER_OBJ_ID, QUIESCER_AUTH_ID, QUIESCER_AGENT_ID,
   QUIESCER_STATE, DBPARTITIONNUM
   FROM TABLE( SYSPROC.SNAP_GET_TBSP_QUIESCER( '', 1)) AS T
```

The following is an example of output from this query.

```
TBSP_NAME  QUIESCER_TS_ID     QUIESCER_OBJ_ID     QUIESCER_AUTH_ID     ...
---------- --------------...- ---------------...- ----------------...- ...
USERSPACE1                 2                   5 SWALKTY              ...

  1 record(s) selected.
```

Output from this query (continued).

```
... QUIESCER_AGENT_ID    QUIESCER_STATE DBPARTITIONNUM
... -------------------- -------------- --------------
...                65983 EXCLUSIVE                   1
```

Information returned

Table 225. Information returned by the SNAPTBSP_QUIESCER administrative view and the SNAP_GET_TBSP_QUIESCER table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
QUIESCER_TS_ID	BIGINT	quiescer_ts_id - Quiescer table space identification
QUIESCER_OBJ_ID	BIGINT	quiescer_obj_id - Quiescer object identification
QUIESCER_AUTH_ID	VARCHAR(128)	quiescer_auth_id - Quiescer user authorization identification
QUIESCER_AGENT_ID	BIGINT	quiescer_agent_id - Quiescer agent identification
QUIESCER_STATE	VARCHAR(14)	quiescer_state - Quiescer state. This interface returns a text identifier based on defines in `sqlutil.h` and is one of: • EXCLUSIVE • UPDATE • SHARE
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPTBSP_RANGE administrative view and SNAP_GET_TBSP_RANGE table function - Retrieve range snapshot information

The SNAPTBSP_RANGE administrative view and the SNAP_GET_TBSP_RANGE table function return information from a range snapshot.

SNAPTBSP_RANGE administrative view

This administrative view allows you to retrieve range snapshot information for the currently connected database.

Used with the SNAPTBSP, SNAPTBSP_PART, SNAPTBSP_QUIESCER and SNAPCONTAINER administrative views, the SNAPTBSP_RANGE administrative view provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 196 on page 704 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPTBSP_RANGE administrative view
- CONTROL privilege on the SNAPTBSP_RANGE administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_TBSP_RANGE table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Select information about table space ranges for all database partitions for the currently connected database.

```
SELECT TBSP_ID, SUBSTR(TBSP_NAME, 1, 15) AS TBSP_NAME, RANGE_NUMBER,
   RANGE_STRIPE_SET_NUMBER, RANGE_OFFSET, RANGE_MAX_PAGE,
   RANGE_MAX_EXTENT, RANGE_START_STRIPE, RANGE_END_STRIPE,
   RANGE_ADJUSTMENT, RANGE_NUM_CONTAINER, RANGE_CONTAINER_ID,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPTBSP_RANGE
   ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TBSP_ID     TBSP_NAME       RANGE_NUMBER     RANGE_STRIPE_SET_NUMBER ...
-------...- --------------- ------------...- ----------------------- ...
          0 SYSCATSPACE                    0                       0 ...
          2 USERSPACE1                     0                       0 ...
          3 SYSTOOLSPACE                   0                       0 ...
          2 USERSPACE1                     0                       0 ...
          2 USERSPACE1                     0                       0 ...

  5 record(s) selected.
```

Output from this query (continued).

```
... RANGE_OFFSET     RANGE_MAX_PAGE       RANGE_MAX_EXTENT     ...
... ------------...- -------------------- -------------------- ...
...                0                11515                 2878 ...
...                0                  479                   14 ...
...                0                  251                   62 ...
...                0                  479                   14 ...
...                0                  479                   14 ...
```

Output from this query (continued).

```
... RANGE_START_STRIPE   RANGE_END_STRIPE     RANGE_ADJUSTMENT     ...
... -------------------- -------------------- -------------------- ...
...                    0                 2878                    0 ...
...                    0                   14                    0 ...
...                    0                   62                    0 ...
...                    0                   14                    0 ...
...                    0                   14                    0 ...
```

Output from this query (continued).

```
... RANGE_NUM_CONTAINER  RANGE_CONTAINER_ID   DBPARTITIONNUM
... -------------------- -------------------- --------------
...                    1                    0              0
...                    1                    0              0
...                    1                    0              0
...                    1                    0              1
...                    1                    0              2
```

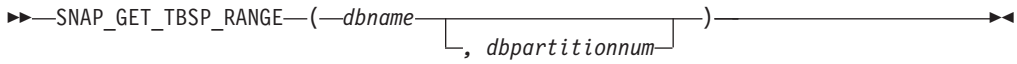
SNAP_GET_TBSP_RANGE table function

The SNAP_GET_TBSP_RANGE table function returns the same information as the SNAPTBSP_RANGE administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_TBSP_V91, SNAP_GET_TBSP_PART_V91, SNAP_GET_TBSP_QUIESCER and SNAP_GET_CONTAINER_V91 table functions, the SNAP_GET_TBSP_RANGE table function provides information equivalent to the GET SNAPSHOT FOR TABLESPACES ON database-alias CLP command.

Refer to Table 196 on page 704 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_TBSP_RANGE table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_TBSP_RANGE table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Select information on the table space range for the table space with tbsp_id = 2 on the currently connected database partition.

```
SELECT TBSP_ID, SUBSTR(TBSP_NAME, 1, 15) AS TBSP_NAME, RANGE_NUMBER,
   RANGE_STRIPE_SET_NUMBER, RANGE_OFFSET, RANGE_MAX_PAGE, RANGE_MAX_EXTENT,
   RANGE_START_STRIPE, RANGE_END_STRIPE, RANGE_ADJUSTMENT,
   RANGE_NUM_CONTAINER, RANGE_CONTAINER_ID
   FROM TABLE(SNAP_GET_TBSP_RANGE('',-1)) AS T WHERE TBSP_ID = 2
```

The following is an example of output from this query.

```
TBSP_ID     TBSP_NAME       RANGE_NUMBER     ...
-------...- --------------- ------------...- ...
          2 USERSPACE1                     0 ...

  1 record(s) selected.
```

Output from this query (continued).

```
... RANGE_STRIPE_SET_NUMBER RANGE_OFFSET     RANGE_MAX_PAGE        ...
... ----------------------- ------------...- --------------...---- ...
...                       0               0                  3967 ...
```

Output from this query (continued).

```
... RANGE_MAX_EXTENT     RANGE_START_STRIPE   RANGE_END_STRIPE     ...
... -------------------- -------------------- -------------------- ...
...                  123                    0                  123 ...
```

Output from this query (continued).

```
... RANGE_ADJUSTMENT     RANGE_NUM_CONTAINER RANGE_CONTAINER_ID
... -------------------- ------------------- --------------------
...                    0                   1                    0
```

Information returned

Table 226. Information returned by the SNAPTBSP_RANGE administrative view and the SNAP_GET_TBSP_RANGE table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_ID	BIGINT	tablespace_id - Table space identification

Table 226. Information returned by the SNAPTBSP_RANGE administrative view and the SNAP_GET_TBSP_RANGE table function (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
RANGE_NUMBER	BIGINT	range_number - Range number
RANGE_STRIPE_SET_NUMBER	BIGINT	range_stripe_set_number - Stripe set number
RANGE_OFFSET	BIGINT	range_offset - Range offset
RANGE_MAX_PAGE	BIGINT	range_max_page_number - Maximum page in range
RANGE_MAX_EXTENT	BIGINT	range_max_extent - Maximum extent in range
RANGE_START_STRIPE	BIGINT	range_start_stripe - Start stripe
RANGE_END_STRIPE	BIGINT	range_end_stripe - End stripe
RANGE_ADJUSTMENT	BIGINT	range_adjustment - Range adjustment
RANGE_NUM_CONTAINER	BIGINT	range_num_containers - Number of containers in range
RANGE_CONTAINER_ID	BIGINT	range_container_id - Range container
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPUTIL administrative view and SNAP_GET_UTIL table function - Retrieve utility_info logical data group snapshot information

The SNAPUTIL administrative view and the SNAP_GET_UTIL table function return snapshot information on utilities from the utility_info logical data group.

SNAPUTIL administrative view

Used in conjunction with the SNAPUTIL_PROGRESS administrative view, the SNAPUTIL administrative view provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

The schema is SYSIBMADM.

Refer to Table 197 on page 708 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPUTIL administrative view
- CONTROL privilege on the SNAPUTIL administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_UTIL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of utilities and their states on all database partitions for all active databases in the instance that contains the connected database.

```
SELECT UTILITY_TYPE, UTILITY_PRIORITY, SUBSTR(UTILITY_DESCRIPTION, 1, 72)
   AS UTILITY_DESCRIPTION, SUBSTR(UTILITY_DBNAME, 1, 17) AS
   UTILITY_DBNAME, UTILITY_STATE, UTILITY_INVOKER_TYPE, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPUTIL ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
UTILITY_TYPE     UTILITY_PRIORITY ...
------------...- ---------------- ...
LOAD                            - ...
LOAD                            - ...
LOAD                            - ...

3 record(s) selected.
```

Output from this query (continued).

```
... UTILITY_DESCRIPTION                                                  ...
... ------------------------------------------------------------------...
... ONLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO TEST .LOADTEST ...
... ONLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO TEST .LOADTEST ...
... ONLINE LOAD DEL AUTOMATIC INDEXING INSERT COPY NO TEST .LOADTEST ...
```

Output from this query (continued).

```
... UTILITY_DBNAME    UTILITY_STATE UTILITY_INVOKER_TYPE DBPARTITIONNUM
... ----------------- ------------- -------------------- --------------
... SAMPLE            EXECUTE       USER                            0
... SAMPLE            EXECUTE       USER                            1
... SAMPLE            EXECUTE       USER                            2
```

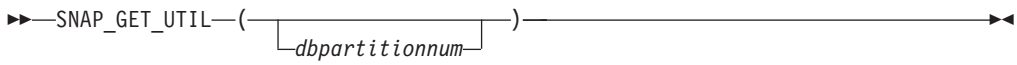
SNAP_GET_UTIL table function

The SNAP_GET_UTIL table function returns the same information as the SNAPUTIL administrative view, but allows you to retrieve the information for a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_UTIL_PROGRESS table function, the SNAP_GET_UTIL table function provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

Refer to Table 197 on page 708 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_UTIL table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_UTIL table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve a list of utility ids with their type and state for the currently connected database partition on database SAMPLE.

```
SELECT UTILITY_ID, UTILITY_TYPE, STATE
   FROM TABLE(SNAP_GET_UTIL(-1)) AS T WHERE UTILITY_DBNAME='SAMPLE'
```

The following is an example of output from this query.

```
UTILITY_ID           UTILITY_TYPE               STATE
-------------------- -------------------------- --------
                   1 BACKUP                     EXECUTE

  1 record(s) selected.
```

Information returned

Table 227. Information returned by the SNAPUTIL administrative view and the SNAP_GET_UTIL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
UTILITY_ID	INTEGER	utility_id - Utility ID. Unique to a database partition.
UTILITY_TYPE	VARCHAR(26)	utility_type - Utility type. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • ASYNC_INDEX_CLEANUP • BACKUP • CRASH_RECOVERY • LOAD • REBALANCE • REDISTRIBUTE • REORG • RESTART_RECREATE_INDEX • RESTORE • ROLLFORWARD_RECOVERY • RUNSTATS
UTILITY_PRIORITY	INTEGER	utility_priority - Utility priority. Priority if utility supports throttling, otherwise null.
UTILITY_DESCRIPTION	VARCHAR(2048)	utility_description - Utility description. Can be null.
UTILITY_DBNAME	VARCHAR(128)	utility_dbname - Database operated on by utility
UTILITY_START_TIME	TIMESTAMP	utility_start_time - Utility start time
UTILITY_STATE	VARCHAR(10)	utility_state - Utility state. This interface returns a text identifier based on the defines in sqlmon.h and is one of: • ERROR • EXECUTE • WAIT
UTILITY_INVOKER_TYPE	VARCHAR(10)	utility_invoker_type - Utility invoker type. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTO • USER
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
PROGRESS_LIST_ATTR	VARCHAR(10)	progress_list_attr - Current progress list attributes

Table 227. Information returned by the SNAPUTIL administrative view and the SNAP_GET_UTIL table function (continued)

Column name	Data type	Description or corresponding monitor element
PROGRESS_LIST_CUR_SEQ_NUM	INTEGER	progress_list_current_seq_num - Current progress list sequence number

SNAPUTIL_PROGRESS administrative view and SNAP_GET_UTIL_PROGRESS table function - Retrieve progress logical data group snapshot information

The SNAPUTIL_PROGRESS administrative view and the SNAP_GET_UTIL_PROGRESS table function return snapshot information about utility progress, in particular, the progress logical data group.

SNAPUTIL_PROGRESS administrative view

Used in conjunction with the SNAPUTIL administrative view, the SNAPUTIL_PROGRESS administrative view provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

The schema is SYSIBMADM.

Refer to Table 198 on page 711 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPUTIL_PROGRESS administrative view
- CONTROL privilege on the SNAPUTIL_PROGRESS administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_UTIL_PROGRESS table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on total and completed units of progress by utility ID.

```
SELECT SELECT UTILITY_ID, PROGRESS_TOTAL_UNITS, PROGRESS_COMPLETED_UNITS,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPUTIL_PROGRESS
```

The following is an example of output from this query.

```
UTILITY_ID PROGRESS_TOTAL_UNITS PROGRESS_COMPLETED_UNITS DBPARTITIONNU
---------- -------------------- ------------------------ ------------
         7                   10                        5            0
         9                   10                        5            1

  1 record(s) selected.
```

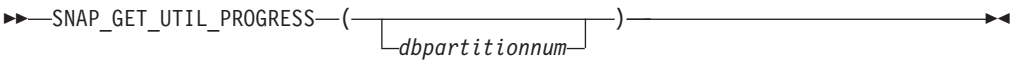
SNAP_GET_UTIL_PROGRESS table function

The SNAP_GET_UTIL_PROGRESS table function returns the same information as the SNAPUTIL_PROGRESS administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used in conjunction with the SNAP_GET_UTIL table function, the SNAP_GET_UTIL_PROGRESS table function provides the same information as the LIST UTILITIES SHOW DETAIL CLP command.

Refer to Table 198 on page 711 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_UTIL_PROGRESS table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_UTIL_PROGRESS table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON

- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve details on the progress of utilities on the currently connect partition.

```
SELECT UTILITY_ID, PROGRESS_TOTAL_UNITS, PROGRESS_COMPLETED_UNITS,
   DBPARTITIONNUM FROM TABLE(SNAP_GET_UTIL_PROGRESS(-1)) as T
```

The following is an example of output from this query.

```
UTILITY_ID PROGRESS_TOTAL_UNITS PROGRESS_COMPLETED_UNITS DBPARTITIONNUM
---------- -------------------- ------------------------ --------------
         7                   10                        5              0

  1 record(s) selected.
```

Information returned

Table 228. Information returned by the SNAPUTIL_PROGRESS administrative view and the SNAP_GET_UTIL_PROGRESS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
UTILITY_ID	INTEGER	utility_id - Utility ID. Unique to a database partition.
PROGRESS_SEQ_NUM	INTEGER	progress_seq_num - Progress sequence number. If serial, the number of the phase. If concurrent, then could be NULL.
UTILITY_STATE	VARCHAR(16)	utility_state - Utility state. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • ERROR • EXECUTE • WAIT
PROGRESS_DESCRIPTION	VARCHAR(2048)	progress_description - Progress description
PROGRESS_START_TIME	TIMESTAMP	progress_start_time - Progress start time. Start time if the phase has started, otherwise NULL.

Table 228. Information returned by the SNAPUTIL_PROGRESS administrative view and the SNAP_GET_UTIL_PROGRESS table function (continued)

Column name	Data type	Description or corresponding monitor element
PROGRESS_WORK_METRIC	VARCHAR(16)	progress_work_metric - Progress work metric. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • NOT_SUPPORT • BYTES • EXTENTS • INDEXES • PAGES • ROWS • TABLES
PROGRESS_TOTAL_UNITS	BIGINT	progress_total_units - Total progress work units
PROGRESS_COMPLETED_UNITS	BIGINT	progress_completed_units - Completed progress work units
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAP_WRITE_FILE procedure

The SNAP_WRITE_FILE procedure writes system snapshot data to a file in the `tmp` subdirectory of the instance directory.

Syntax

The schema is SYSPROC.

Procedure parameters

requestType
An input argument of type VARCHAR (32) that specifies a valid snapshot request type. The possible request types are text identifiers based on defines in `sqlmon.h`, and are one of:

- APPL_ALL
- BUFFERPOOLS_ALL
- DB2
- DBASE_ALL
- DBASE_LOCKS
- DBASE_TABLES
- DBASE_TABLESPACES
- DYNAMIC_SQL

dbname
An input argument of type VARCHAR(128) that specifies a valid database

name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify NULL or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If a null value is specified, -1 is set implicitly.

Authorization

To execute the procedure, a user must have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority. The saved snapshot can be read by users who do not have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority by passing null values as the inputs to snapshot table functions.

Example

Take a snapshot of database manager information by specifying a request type of 'DB2' (which corresponds to SQLMA_DB2), and defaulting to the currently connected database and current database partition.

```
CALL SYSPROC.SNAP_WRITE_FILE ('DB2', '', -1)
```

This will result in snapshot data being written to the instance temporary directory, which is sqllib/tmp/SQLMA_DB2.dat on UNIX operating systems, and sqllib\DB2\tmp\SQLMA_DB2.dat on a Windows operating system.

Usage notes

If an unrecognized input parameter is provided, the following error is returned: SQL2032N The "REQUEST_TYPE" parameter is not valid.

TBSP_UTILIZATION administrative view - Retrieve table space configuration and utilization information

The TBSP_UTILIZATION administrative view returns table space configuration and utilization information. It retrieve a similar report to the LIST TABLESPACES command on a single partitioned database. Its information is based on the SNAPTBSP, SNAPTBSP_PART administrative views and TABLESPACES catalog view.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the TBSP_UTILIZATION administrative view
- CONTROL privilege on the TBSP_UTILIZATION administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve the same report as the LIST TABLESPACES command on a single partitioned database.

```
SELECT TBSP_ID, SUBSTR(TBSP_NAME,1,20) as TBSP_NAME, TBSP_TYPE,
   TBSP_CONTENT_TYPE, TBSP_STATE FROM SYSIBMADM.TBSP_UTILIZATION
```

The following is an example of output for this query.

```
TBSP_ID     TBSP_NAME            TBSP_TYPE  ...
--------...- -------------------- ---------- ...
          0 SYSCATSPACE          SMS        ...
          1 TEMPSPACE1           SMS        ...
          2 USERSPACE1           SMS        ...
          3 SYSTOOLSPACE         SMS        ...
          4 SYSTOOLSTMPSPACE     SMS        ...
```

Output for this query (continued).

```
... TBSP_CONTENT_TYPE TBSP_STATE
... ----------------- -----------
... ANY               NORMAL
... SYSTEMP           NORMAL
... ANY               NORMAL
... ANY               NORMAL
... USRTEMP           NORMAL
```

Information returned

Table 229. Information returned by the TBSP_UTILIZATION administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_ID	BIGINT	tablespace_id - Table space identification
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_TYPE	VARCHAR(10)	tablespace_type - Table space type
TBSP_CONTENT_TYPE	VARCHAR(10)	tablespace_content_type - Table space content type
TBSP_CREATE_TIME	TIMESTAMP	Creation time of the table space.
TBSP_STATE	VARCHAR(256)	tablespace_state - Table space state
TBSP_TOTAL_SIZE_KB	BIGINT	The total size of the table space in KB, calculated as total_pages*pagesize/1024.
TBSP_USABLE_SIZE_KB	BIGINT	The total usable size of the table space in KB, calculated as usable_pages*pagesize/1024.

Table 229. Information returned by the TBSP_UTILIZATION administrative view (continued)

Column name	Data type	Description or corresponding monitor element
TBSP_USED_SIZE_KB	BIGINT	The total used size of the table space in KB, calculated as used_pages*pagesize/1024.
TBSP_FREE_SIZE_KB	BIGINT	The total available size of the table space in KB, calculated as free_pages*pagesize/1024.
TBSP_UTILIZATION_PERCENT	BIGINT	The utilization of the table space as a percentage. Calculated as (used_pages/usable_pages)*100, if usable_pages is available. Otherwise, -1 will be displayed.
TBSP_TOTAL_PAGES	BIGINT	tablespace_total_pages - Total pages in table space
TBSP_USABLE_PAGES	BIGINT	tablespace_usable_pages - Usable pages in table space
TBSP_USED_PAGES	BIGINT	tablespace_used_pages - Used pages in table space
TBSP_FREE_PAGES	BIGINT	tablespace_free_pages - Free pages in table space
TBSP_PAGE_TOP	BIGINT	tablespace_page_top - Table space high water mark
TBSP_PAGE_SIZE	INTEGER	tablespace_page_size - Table space page size
TBSP_EXTENT_SIZE	INTEGER	tablespace_extent_size - Table space extent size
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size
TBSP_MAX_SIZE	BIGINT	tablespace_max_size - Maximum table space size
TBSP_INCREASE_SIZE	BIGINT	tablespace_increase_size - Increase size in bytes
TBSP_INCREASE_SIZE_PERCENT	SMALLINT	tablespace_increase_size_percent - Increase size by percent
TBSP_LAST_RESIZE_TIME	TIMESTAMP	tablespace_last_resize_time - Time of last successful resize
TBSP_LAST_RESIZE_FAILED	SMALLINT	tablespace_last_resize_failed - Last resize attempt failed
TBSP_USING_AUTO_STORAGE	SMALLINT	tablespace_using_auto_storage - Table space enabled for automatic storage
TBSP_AUTO_RESIZE_ENABLED	SMALLINT	tablespace_auto_resize_enabled - Table space automatic resizing enabled
DBPGNAME	VARCHAR(128)	Name of the database partition group for the table space.
TBSP_NUM_CONTAINERS	BIGINT	tablespace_num_containers - Number of containers in table space

Table 229. Information returned by the TBSP_UTILIZATION administrative view (continued)

Column name	Data type	Description or corresponding monitor element
REMARKS	VARCHAR(254)	User-provided comment.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

TOP_DYNAMIC_SQL administrative view - Retrieve information on the top dynamic SQL statements

The TOP_DYNAMIC_SQL administrative view returns the top dynamic SQL statements sortable by number of executions, average execution time, number of sorts, or sorts per statement. These are the queries that should get focus to ensure they are well tuned.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the TOP_DYNAMIC_SQL administrative view
- CONTROL privilege on the TOP_DYNAMIC_SQL administrative view
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Identify the top 5 most frequently run SQL.

```
SELECT NUM_EXECUTIONS, AVERAGE_EXECUTION_TIME_S, STMT_SORTS,
   SORTS_PER_EXECUTION, SUBSTR(STMT_TEXT,1,60) AS STMT_TEXT
   FROM SYSIBMADM.TOP_DYNAMIC_SQL
   ORDER BY NUM_EXECUTIONS DESC FETCH FIRST 5 ROWS ONLY
```

The following is an example of output for this query.

```
NUM_EXECUTIONS       AVERAGE_EXECUTION_TIME_S STMT_SORTS           ...
-------------------- ------------------------ -------------------- ...
                 148                        0                    0 ...
                 123                        0                    0 ...
                   2                        0                    0 ...
                   1                        0                    0 ...
                   1                        0                    0 ...

  5 record(s) selected.
```

Output for this query (continued).

```
... SORTS_PER_EXECUTION  ...
... -------------------- ...
...                    0 ...
...                    0 ...
...                    0 ...
...                    0 ...
...                    0 ...
```

Output for this query (continued).

```
... STMT_TEXT
... ------------------------------------------------------------
... SELECT A.ID, B.EMPNO, B.FIRSTNME, B.LASTNAME, A.DEPT FROM E
... SELECT A.EMPNO, A.FIRSTNME, A.LASTNAME, B.LOCATION, B.MGRNO
... SELECT A.EMPNO, A.FIRSTNME, A.LASTNAME, B.DEPTNAME FROM EMP
... SELECT ATM.SCHEMA, ATM.NAME, ATM.CREATE_TIME, ATM.LAST_WAIT,
... SELECT * FROM JESSICAE.EMP_RESUME
```

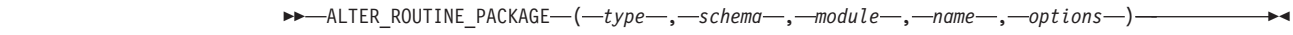
Information returned

Table 230. Information returned by the TOP_DYNAMIC_SQL administrative view

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	Timestamp for the report.
NUM_EXECUTIONS	BIGINT	num_compilations - Statement compilations
AVERAGE_EXECUTION_TIME_S	BIGINT	Average execution time.
STMT_SORTS	BIGINT	stmt_sorts - Statement sorts
SORTS_PER_EXECUTION	BIGINT	Number of sorts per statement execution.
STMT_TEXT	CLOB(2 M)	stmt_text - SQL statement text
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

Chapter 16. SQL procedures routines

ALTER_ROUTINE_PACKAGE procedure



The schema is SYSPROC.

This procedure alters values for the package associated with a compiled SQL routine or a compiled trigger, without the need for rebinding. It is functionally equivalent to the ALTER PACKAGE command, except that it takes an object name instead of a package name as an argument. The ALTER_ROUTINE_PACKAGE procedure can be invoked from the command line or called from an application.

type
An input argument of type CHAR(2) that specifies the type of routine or compiled trigger, using one of the following values:

- 'P ' for a procedure
- 'SP' for the specific name of a procedure
- 'F' for a compiled function
- 'SF' for a specific name of a compiled function
- 'T' for a compiled trigger

schema
An optional input argument of type VARCHAR(128), which specifies the schema of the routine or trigger. If a schema is not specified, the value will default to the value of the CURRENT SCHEMA special register. This parameter is case sensitive.

module
An optional input argument of type VARCHAR(128), which specifies the name of the module where the routine resides. This parameter cannot be specified for triggers. If this parameter is not specified, then module routines are ignored. This parameter is case sensitive.

name
An input argument of type VARCHAR(128), which specifies the name of the routine or trigger. This parameter is case sensitive.

options
An input argument of type VARCHAR(1024), which specifies a list of any options supported by the ALTER PACKAGE statement. At least one ALTER PACKAGE clause must be supplied within the *options* parameter.

Examples

Alter the underlying package for an existing stored procedure by the name of UPDATE_EMPLOYEE.

```
CALL SYSPROC.ALTER_ROUTINE_PACKAGE ('P','','','UPDATE_EMPLOYEE',
   'ACCESS PLAN REUSE YES OPTIMIZATION PROFILE AYYANG.INDEXHINTS')
```

Alter the package for a compiled trigger called MIN_SALARY, in the DRICARD schema.

```
CALL SYSPROC.ALTER_ROUTINE_PACKAGE ('T','DRICARD','','MIN_SALARY',
   'OPTIMIZATION PROFILE AYYANG.INDEXHINTS')
```

Alter the package for a compiled function, using a three part name.

```
CALL SYSPROC.ALTER_ROUTINE_PACKAGE ('F','DRICARD','MODULE','FUNCTION','APREUSE YES')
```

GET_ROUTINE_OPTS

The schema is SYSPROC.

The GET_ROUTINE_OPTS function returns a character string value of the options that are to be used for the creation of SQL procedures in the current session.

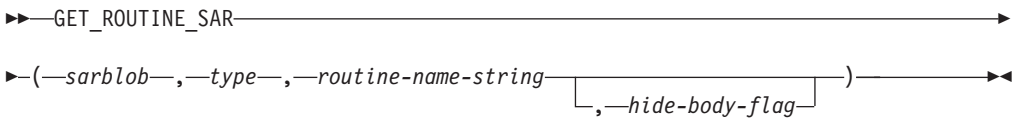
The result of the function is a varying-length character string (VARCHAR) value with a length attribute of 1024.

Example:

Return the options to be used for the creation of SQL procedures as the result of a query.

```
SELECT GET_ROUTINE_OPTS()
  FROM SYSIBM.SYSDUMMY1
```

GET_ROUTINE_SAR



The schema is SYSFUN.

The GET_ROUTINE_SAR procedure retrieves the necessary information to install the same routine in another database server running the same level on the same operating system. The information is retrieved into a single BLOB string representing an SQL archive file.

Authorization

EXECUTE privilege on GET_ROUTINE_SAR procedure

sarblob
An output argument of type BLOB(3M) that contains the routine SAR file contents.

type
An input argument of type CHAR(2) that specifies the type of routine, using one of the following values:

- 'P ' for a procedure
- 'SP' for the specific name of a procedure

routine-name-string
An input argument of type VARCHAR(257) that specifies a qualified name of the routine. If no schema name is specified, the default is the CURRENT SCHEMA when the routine is processed. The *routine-name-string* cannot include double quotation marks (").

hide-body-flag
An input argument of type INTEGER that specifies (using one of the following values) whether or not the routine body should be hidden when the routine text is extracted from the catalogs. Valid values are:

0 Leave the routine text intact. This is the default value.

1 Replace the routine body with an empty body when the routine text is extracted from the catalogs.

The qualified name of the routine is used to determine which routine to retrieve. The routine that is found must be an SQL routine. Not using a specific name may result in more than one routine, and an error is raised (SQLSTATE 42725). If this occurs, the specific name of the desired routine must be used.

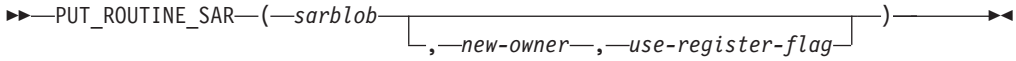
The SAR file must include a bind file, which may not be available at the server. If the bind file cannot be found and stored in the SAR file, an error is raised (SQLSTATE 55045).

PUT_ROUTINE_SAR

The PUT_ROUTINE_SAR procedure passes the necessary file to create an SQL routine at the server and then defines the routine.

Authorization

DBADM



The schema is SYSFUN.

sarblob
An input argument of type BLOB(3M) that contains the routine SAR file contents.

new-owner
An input argument of type VARCHAR(128) that contains an authorization-name used for authorization checking of the routine. The *new-owner* must have the necessary privileges for the routine to be defined. If *new-owner* is not specified, the authorization-name of the original routine definer is used.

use-register-flag
An input argument of type INTEGER that indicates whether or not the CURRENT SCHEMA and CURRENT PATH special registers are used to define the routine. If the special registers are not used, the settings for the default schema and SQL path are the settings used when the routine was originally defined. Possible values for *use-register-flag*:

0 Do not use the special registers of the current environment

1 Use the CURRENT SCHEMA and CURRENT PATH special registers.

If the value is 1, CURRENT SCHEMA is used for unqualified object names in the routine definition (including the name of the routine) and CURRENT PATH is used to resolve unqualified routines and data types in the routine definition. If the *use-registers-flag* is not specified, the behavior is the same as if a value of 0 was specified.

The identification information contained in *sarblob* is checked to confirm that the inputs are appropriate for the environment, otherwise an error is raised (SQLSTATE 55046). The PUT_ROUTINE_SAR procedure then uses the contents of the *sarblob* to define the routine at the server.

The contents of the *sarblob* argument are extracted into the separate files that make up the SQL archive file. The shared library and bind files are written to files in a temporary directory. The environment is set so that the routine definition statement processing is aware that compiling and linking are not required, and that the location of the shared library and bind files is available. The contents of the DDL file are then used to dynamically execute the routine definition statement.

No more than one procedure can be concurrently installed under a given schema.

Processing of this statement might result in the same errors as executing the routine definition statement using other interfaces. During routine definition processing, the presence of the shared library and bind files is noted and the precompile, compile and link steps are skipped. The bind file is used during bind processing and the contents of both files are copied to the usual directory for an SQL routine.

If a GET ROUTINE or a PUT ROUTINE operation (or their corresponding procedure) fails to execute successfully, it will always return an error (SQLSTATE 38000), along with diagnostic text providing information about the cause of the failure. For example, if the procedure name provided to GET ROUTINE does not identify an SQL procedure, diagnostic "-204, 42704" text will be returned, where "-204" and "42704" are the SQLCODE and SQLSTATE, respectively, that identify the cause of the problem. The SQLCODE and SQLSTATE in this example indicate that the procedure name provided in the GET ROUTINE command is undefined.

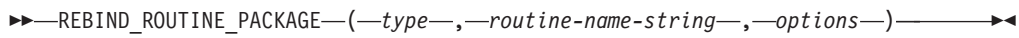
REBIND_ROUTINE_PACKAGE procedure - rebind a package

The REBIND_ROUTINE_PACKAGE procedure rebinds the package associated with an SQL procedure, routine, compiled function, or trigger. It is functionally equivalent to the REBIND command, except that it takes a procedure name, instead of a package name, as an argument. The REBIND_ROUTINE_PACKAGE procedure can be invoked from the command line or called from an application.

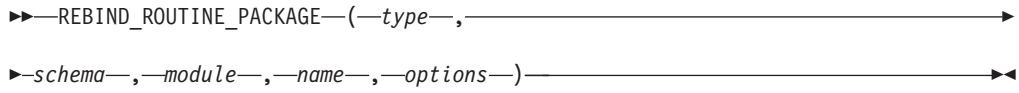
Syntax

There are two equally valid methods to invoke REBIND_ROUTINE_PACKAGE. The only difference between the two invocations is the method of specifying the routine name. In the first instance, the *routine-name-string* variable consists of identifier names separated by periods. In the second method, the routine is identified by separate values for each of the *schema*, *module* and *name* values.

Method 1:



Method 2:



The schema is SYSPROC.

Procedure parameters

type
An input argument of type CHAR(2) that specifies the type of routine or compiled trigger, using one of the following values:

- 'P ' for a procedure
- 'SP' for the specific name of a procedure
- 'F' for a compiled function
- 'SF' for a specific name of a compiled function
- 'T' for a compiled trigger

routine-name-string **(method 1 only)**
An input argument of type VARCHAR(386) which specifies the name of the routine or trigger. Trigger names consist of two parts separated by a period and are in the format schema.trigger where the schema is optional. Routine names consist of three part names separated by periods and are in the format schema.module.routine where schema and module are optional. If schema is not specified, the value defaults to the value of the CURRENT SCHEMA special register. If a two-part name is specified, the first part is initially interpreted as a schema name; if the routine is not found under that schema, the first part is interpreted as a module name, and an attempt is made to find the routine in a module of that name under the CURRENT SCHEMA. The schema, module or object names cannot include double quotation marks (") or periods(.).

schema **(method 2 only)**
An optional input argument of type VARCHAR(128) that specifies the schema of the routine or trigger. If a schema is not specified, the value will default to the value of the CURRENT SCHEMA special register. This parameter is case sensitive.

module **(method 2 only)**
An optional input argument of type VARCHAR(128) that specifies the name of the module where the routine resides. Do not specify this parameter for triggers. Module routines are ignored if this parameter is not specified. This parameter is case sensitive.

name **(method 2 only)**
An input argument of type VARCHAR(128) that specifies the name of the routine or trigger. This parameter is case sensitive.

options
An optional input argument of type VARCHAR(1024) which specifies any list of rebind options following the REBIND command syntax. A single value of “ANY” or “CONSERVATIVE” is also supported for backward compatibility and is interpreted as the value for the RESOLVE rebind option.

The qualified name of the routine is used to determine which routine to retrieve. The routine that is found must be an SQL routine; otherwise, an error is returned (SQLSTATE 428F7). If a specific name is not used, more than one routine may be found, and an error is returned (SQLSTATE 42725). If this occurs, the specific name of the desired routine must be used.

Examples

Example 1: Rebind the package of routine UPDATE_EMPLOYEE using the RESOLVE, REOPT and APREUSE options.

```
Method 1:
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'P','UPDATE_EMPLOYEE','RESOLVE ANY REOPT ONCE APREUSE YES')
Method 2:
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'P','','','UPDATE_EMPLOYEE','RESOLVE ANY REOPT ONCE APREUSE YES')
```

Example 2: Rebind the package of routine UPDATE_EMPLOYEE with no options.

```
Method 1:
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'P','UPDATE_EMPLOYEE','')
Method 2:
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'P','','','UPDATE_EMPLOYEE','')
```

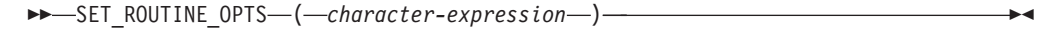
Example 3: Rebind the package of a compiled trigger.

```
Method 1:
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'T','DRICARD.MIN_SALARY','REOPT ALWAYS')
Method 2:
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'T','DRICARD','','MIN_SALARY','REOPT ALWAYS')
```

Example 4: Rebind the package of a compiled function using a three part name.

```
Method 1
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'F','DRICARD.MODULE.FUNCTION','REOPT ALWAYS')
Method 2
CALL SYSPROC.REBIND_ROUTINE_PACKAGE (
  'F','DRICARD','MODULE','FUNCTION','REOPT ALWAYS')
```

SET_ROUTINE_OPTS



The schema is SYSPROC.

The SET_ROUTINE_OPTS procedure sets the options that are to be used for the creation of SQL procedures in the current session. This setting overrides the instance-wide setting specified in the DB2_SQLROUTINE_PREPOPTS registry variable.

character-expression
An input argument of type VARCHAR(1024) that specifies the options setting for the current session.

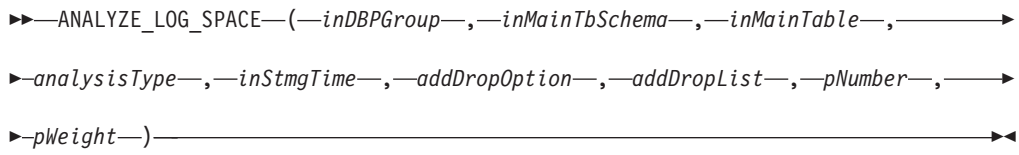
Specified options are valid for the duration of the session. If the null value is specified as the argument, the value of the DB2_SQLROUTINE_PREPOPTS registry variable is restored as the default options setting for the current session. For a list of the allowed options, see the description of the DB2_SQLROUTINE_PREPOPTS registry variable under “Query compiler variables”.

Chapter 17. Stepwise redistribute routines

ANALYZE_LOG_SPACE procedure - Retrieve log space analysis information

The ANALYZE_LOG_SPACE procedure returns the log space analysis results for each of the database partitions of the given database partition group.

Syntax



The schema is SYSPROC.

Procedure parameters

inDBPGroup
An input argument of type VARCHAR (128) that specifies the database partition group name.

inMainTbSchema
An input argument of type VARCHAR (128) that specifies the schema of the main table

inMainTable
An input argument of type VARCHAR (128) that specifies the main table within the database partition group, usually the largest table in the database partition group.

analysisType
An input argument of type SMALLINT that specifies an indicator for analysis type:

- SWRD_USE_STMG_TABLE (1): indicates that the information in the storage management tables is used to find the table row count per database partition. This type should only be used if the storage management tables are setup, and at least one storage snapshot has been taken for the database partition group that is to be redistributed.
- SWRD_USE_REALTIME_ANALYSIS (2): indicates that a SELECT query is used to find the table row count per database partition.

inStmgTime
An input argument of type VARCHAR (26) that specifies the timestamp for the storage management record. This parameter is ignored when *analysisType* is set to SWRD_USE_REALTIME_ANALYSIS.

addDropOption
An input argument of type CHAR (1) that specifies database partitions are being added or dropped:

- 'A': Adding database partitions.

- 'D': Dropping database partitions.
- 'N': No adding or dropping.

addDropList
An input argument of type VARCHAR (6000) that specifies the database partitions to be added or dropped. This database partition numbers are specified in a comma-separated string format and no spaces are allowed in the string.

pNumber
An input argument of type VARCHAR (6000) that specifies all the database partition numbers corresponding to the database partition weight. Each database partition number is between 0 and 999, and the database partition numbers are specified in a comma-separated string with no spaces in the string.

pWeight
An input argument of type VARCHAR (6000) that specifies all the database partition weights that the user has specified corresponding to the database partition numbers in the *pNumber* string. Each database partition weight is a number between 0 and 32767, and database partition weights are specified in a comma-separated string with no spaces in the string.

Authorization

- SYSADM, SYSMON, SYSCTRL, or SYSMAINT
- EXECUTE privilege on the ANALYZE_LOG_SPACE procedure

Example

Analyze the effect of adding a database partition without applying the changes. In the following case, the hypothesis is adding database partition 40, 50 and 60 to the database partition group, and for database partitions 10,20,30,40,50,60, using a respective target ratio of 1:2:1:2:1:2. Note that in this example, only partitions 10, 20 and 30 actually exist in the database partition group

```
CALL SYSPROC.ANALYZE_LOG_SPACE('IBMDEFAULTGROUP', 'TEST',
   'EMP', 2, ' ', 'A', '40,50,60', '10,20,30,40,50,60',
   '1,2,1,2,1,2')
```

Analyze the effect of dropping a database partition without applying the changes. In the following case, the hypothesis is dropping database partition 30 from the database partition group, and redistributing the data in database partitions 10 and 20 using a respective target ratio of 1 : 1. Note that in this example, all database partitions 10, 20 and 30 should exist in the database partition group

```
CALL SYSPROC.ANALYZE_LOG_SPACE('IBMDEFAULTGROUP', 'TEST',
   'EMP', 2, ' ', 'D', '30', '10,20','1,1')
```

Usage notes

"-1" is used as an output value for parameters when their values cannot be obtained.

The redistribute stored procedures and functions work only in partitioned database environments, where a distribution key has been defined for each table.

Information returned

The ANALYZE_LOG_SPACE procedure returns a result set (an open cursor) of the log space analysis results, containing the following fields for each of the database partitions of the given database partition group.

Table 231. Information returned by the ANALYZE_LOG_SPACE procedure

Column name	Column type	Description
PARTITION_NUM	SMALLINT	The database partition number of the log space analysis.
TOTAL_LOG_SIZE	BIGINT	Total log space allocated in bytes, -1 indicates unlimited size.
AVAIL_LOG_SPACE	BIGINT	The amount of log space in bytes that is free and can be used by the redistribute process.
DATA_SKEW	BIGINT	The absolute value in bytes of the size of data which is deviated from the target level.
REQ_LOG_SPACE	BIGINT	The amount of space in bytes required to reach the desired data distribution.
NUM_OF_STEPS	SMALLINT	The number of steps needed to reduce the data skew to zero.
MAX_STEP_SIZE	BIGINT	The maximum amount of data in bytes that can be moved at a time, without causing a log full error.

GENERATE_DISTFILE procedure - Generate a data distribution file

The GENERATE_DISTFILE procedure generates a data distribution file for the given table and saves it under the given fileName.

Syntax

```
►►—GENERATE_DISTFILE—(—inTbSchema—,—inTbName—,—fileName—)————————————►◄
```

The schema is SYSPROC.

Procedure parameters

inTbSchema
An input argument of type VARCHAR (128) that specifies the table schema name.

inTbName
An input argument of type VARCHAR (128) that specifies the table name.

fileName
An input or output argument of type VARCHAR (255) that specifies data distribution file name. If the given file name is just a file name, the file will be saved in the `tmp` sub-directory under the instance directory, and the full file path name will be returned in the parameter.

Authorization

- EXECUTE privilege on the GENERATE_DISTFILE procedure.

- SELECT privilege on SYSCAT.TABLES, SYSCAT.COLUMNS, and the specified table.

In addition, the fenced user ID must be able to create files in the tmp sub-directory under the instance directory.

Example

Generate a data distribution file to be used by the redistribute process.

```
CALL SYSPROC.GENERATE_DISTFILE('TEST', 'EMP',
   '$HOME/sqllib/function/SAMPLE.IBMDEFAULTGROUP_swrdData.dst')"
```

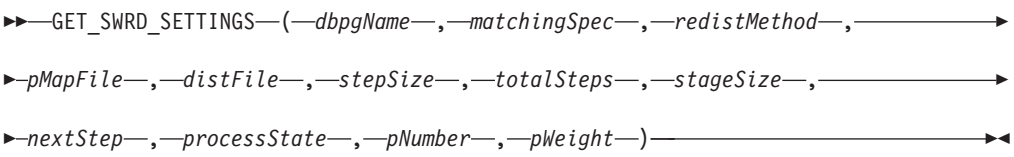
Usage notes

The redistribute stored procedures and functions work only in partitioned database environments, where a distribution key has been defined for each table.

GET_SWRD_SETTINGS procedure - Retrieve redistribute information

The GET_SWRD_SETTINGS procedure reads the existing redistribute registry records for the given database partition group.

Syntax



The schema is SYSPROC.

Procedure parameters

dbpgName
An input argument of type VARCHAR(128) that specifies the database partition group name against which the redistribute process is to run.

matchingSpec
An input argument of type SMALLINT that specifies the bitwise field identifier(s) from Table 232, indicating the target fields to be returned by the output parameters. Those output parameters that are not required can be set to null.

For example, if *matchingSpec* is set to 96, which is the integer value of (REDIST_STAGE_SIZE | REDIST_NEXT_STEP), the caller of this function only needs to provide *stageSize* and *nextStep* to receive the values, and the remainder of the output parameters can be null.

Table 232. Bitwise field identifiers

Field Name	Hexadecimal value	Decimal value
REDIST_METHOD	0x0001<<0	1
REDIST_PMAP_FILE	0x0001<<1	2
REDIST_DIST_FILE	0x0001<<2	4

Table 232. Bitwise field identifiers (continued)

Field Name	Hexadecimal value	Decimal value
REDIST_STEP_SIZE	0x0001<<3	8
REDIST_NUM_STEPS	0x0001<<4	16
REDIST_STAGE_SIZE	0x0001<<5	32
REDIST_NEXT_STEP	0x0001<<6	64
REDIST_PROCESS_STATE	0x0001<<7	128
REDIST_PWEIGHT_START_NODE	0x0001<<8	256
REDIST_PWEIGHT	0x0001<<9	512

redistMethod
An output argument of type SMALLINT that specifies whether the redistribute is to run using the data distribution file or the target distribution map. There are two possible return values:

- 2: indicates that the redistribute process will work with a data distribution file as input.
- 3: indicates that the redistribute process will work with a target distribution map as input.

pMapFile
An output argument of type VARCHAR (255) that specifies the full path file name of the target distribution map on the database server.

distFile
An output argument of type VARCHAR (255) that specifies the full path file name of the data distribution file on the database server.

stepSize
An output argument of type BIGINT that specifies the maximum number of rows that can be moved before a commit must be called to prevent a log full situation. The number can be changed in each redistribution step.

totalSteps
An output argument of type SMALLINT that specifies the number of steps it takes to completely redistribute the given database partition group.

stageSize
An output argument of type SMALLINT that specifies the number of steps to be run consecutively.

nextStep
An output argument of type SMALLINT that specifies the index separating which steps have been completed, and what still needs to be run.

processState
An output argument of type SMALLINT that indicates whether or not the redistribute process will be stopped at the next check point. A check point is placed at beginning of each redistribute step. If this argument is set to 1, the step will not start; if the value is 0, the step will proceed.

pNumber
An output argument of type VARCHAR (6000) that might return a list of comma-separated database partition numbers in a string format. These partition numbers can be either the database partitions that are currently used by the database partition group, or the ones to be added or dropped. The

sequence and the count of these partition numbers correspond to the target partition weight returned by the *pWeight* variable.

pWeight
An output argument of type VARCHAR (6000) that might return a list of comma-separated target database partition weight numbers. The sequence and the count of these partition weights correspond to the partition numbers returned by the *pNumber* variable.

Authorization

EXECUTE privilege on the GET_SWRD_SETTINGS procedure.

Example

Report the content of the step wise redistribution plan for the given database partition group.

```
CALL SYSPROC.GET_SWRD_SETTINGS
   ('IBMDEFAULTGROUP', 255, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

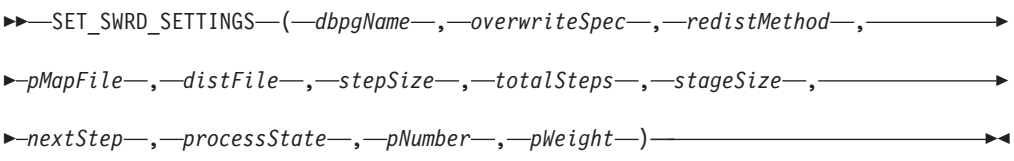
Usage note

The redistribute stored procedures and functions work only in partitioned database environments, where a distribution key has been defined for each table.

SET_SWRD_SETTINGS procedure - Create or change redistribute registry

The SET_SWRD_SETTINGS procedure creates or make changes to the redistribute registry. If the registry does not exist, it creates it and add records into it. If the registry already exists, it uses *overwriteSpec* to identify which of the field values need to be overwritten. The *overwriteSpec* field enables this function to take NULL inputs for the fields that do not need to be updated.

Syntax



The schema is SYSPROC.

Procedure parameters

dbpgName
An input argument of type VARCHAR(128) that specifies the database partition group name against which the redistribute process is to run.

overwriteSpec
Bitwise field identifier(s) from Table 233 on page 873 indicating the target fields to be written or overwritten into the redistribute settings registry.

Table 233. Bitwise field identifiers

Field Name	Hexadecimal value	Decimal value
REDIST_METHOD	0x0001<<0	1
REDIST_PMAP_FILE	0x0001<<1	2
REDIST_DIST_FILE	0x0001<<2	4
REDIST_STEP_SIZE	0x0001<<3	8
REDIST_NUM_STEPS	0x0001<<4	16
REDIST_STAGE_SIZE	0x0001<<5	32
REDIST_NEXT_STEP	0x0001<<6	64
REDIST_PROCESS_STATE	0x0001<<7	128
REDIST_PWEIGHT_START_NODE	0x0001<<8	256
REDIST_PWEIGHT	0x0001<<9	512

redistMethod
An input argument of type SMALLINT that specifies whether the redistribute is to run using the data distribution file or the target distribution map. The two valid input values are:

- 2: indicate that the redistribute process will work with a data distribution file as input.
- 3: indicate that the redistribute process will work with a target distribution map as input.

pMapFile
An input argument of type VARCHAR (255) that specifies the full path file name of the target distribution map on the database server.

distFile
An input argument of type VARCHAR (255) that specifies the full path file name of the data distribution file on the database server..

stepSize
An input argument of type BIGINT that specifies the maximum number of rows that can be moved before a commit must be called to prevent a log full situation. The number can be changed in each redistribution step. The value "-2" can be used for *stepSize* to indicate that the number is unlimited.

totalSteps
An input argument of type SMALLINT that specifies the number of steps it takes to completely redistribute the given database partition group. The value "-2" can be used *totalSteps* to indicate that the number is unlimited.

stageSize
An input argument of type SMALLINT that specifies the number of steps to be run consecutively.

nextStep
An input argument of type SMALLINT that specifies the index separating which steps have been completed, and what still needs to be run.

processState
An input argument of type SMALLINT that indicates whether or not the redistribute process will be stopped at the next check point. A check point is placed at beginning of each redistribute step. If this argument is set to 1, the step will not start; if the value is 0, the step will proceed.

pNumber
An input argument of type VARCHAR (6000) that can contain a list of comma-separated database partition numbers in a string format. These partition numbers can be either the database partitions that are currently used by the database partition group, or the ones to be added or dropped. The sequence and the count of these partition numbers correspond to the target partition weight returned by the *pWeight* variable. Each database partition number is between 0 and 999, and there are no spaces are allowed in the string.

pWeight
An input argument of type VARCHAR (6000) that can contain a comma-separated string of all the database partition weights the user has specified, corresponding to the database partition numbers in the *pNumber* string. Each database partition weight is a number between 0 and 32767, and no spaces are allowed in the string.

Authorization

EXECUTE privilege on the SET_SWRD_SETTINGS procedure.

Example

Write a step wise redistribution plan into a registry. Setting *processState* to 1, might cause a currently running step wise redistribute stored procedure to complete the current step and stop, until this parameter is reset to 0, and the redistribute stored procedure is called again.

```
CALL SYSPROC.SET_SWRD_SETTINGS('IBMDEFAULTGROUP', 255, 0, ' ',
   '$HOME/sqllib/function/TEST.IBMDEFAULTGROUP_swrdData.dst', 1000,
   12, 2, 1, 0, '10,20,30', '50,50,50')
```

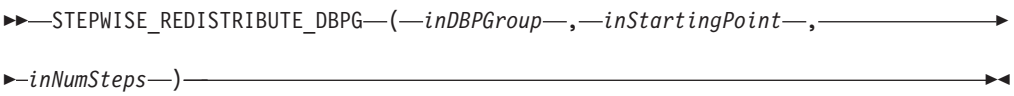
Usage notes

The redistribute stored procedures and functions work only in partitioned database environments, where a distribution key has been defined for each table.

STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of database partition group

The STEPWISE_REDISTRIBUTE_DBPG procedure redistributes part of the database partition group according to the input specified for the procedure, and the setting file created or updated by the SET_SWRD_SETTINGS procedure.

Syntax



The schema is SYSPROC.

Procedure parameters

inDBPGroup
An input argument of type VARCHAR (128) that specifies the name of the target database partition group.

inStartingPoint
An input argument of type SMALLINT that specifies the starting point to use. If the parameter is set to a positive integer and is not NULL, the STEPWISE_REDISTRIBUTE_DBPG procedure uses this value instead of using the *nextStep* value specified in the setting file. This is a useful option when you want to rerun the STEPWISE_REDISTRIBUTE_DBPG procedure from a particular step. If the parameter is set to NULL, the *nextStep* value is used.

inNumSteps
An input argument of type SMALLINT that specifies the number of steps to run. If the parameter is set to a positive integer and is not NULL, the STEPWISE_REDISTRIBUTE_DBPG procedure uses this value instead of using the *stageSize* value specified in the setting file. This is a useful option when you want to rerun the STEPWISE_REDISTRIBUTE_DBPG procedure with a different number of steps than what is specified in the settings. For example, if there are five steps in a scheduled stage, and the redistribution process failed at step 3, the STEPWISE_REDISTRIBUTE_DBPG procedure can be called to run the remaining three steps once the error condition has been corrected. If the parameter is set to NULL, the *stageSize* value is used. The value -2 can be used in this procedure to indicate that the number is unlimited.

Note: There is no parameter for specifying the equivalent of the **NOT ROLLFORWARD RECOVERABLE** option on the REDISTRIBUTE DATABASE PARTITION GROUP command. Logging is always performed for row data redistribution performed when the STEPWISE_REDISTRIBUTE_DBPG procedure is used.

Authorization

- EXECUTE privilege on the STEPWISE_REDISTRIBUTE_DBPG procedure
- SYSADM, SYSCTRL or DBADM

Example

Redistribute the database partition group "IBMDEFAULTGROUP" according to the redistribution plan stored in the registry by the SET_SWRD_SETTINGS procedure. It is starting with step 3 and redistributes the data until 2 steps in the redistribution plan are completed.

```
CALL SYSPROC.STEPWISE_REDISTRIBUTE_DBPG('IBMDEFAULTGROUP', 3, 2)
```

Usage notes

If the registry value for *processState* is updated to 1 using the SET_SWRD_SETTINGS procedure after the STEPWISE_REDISTRIBUTE_DBPG procedure execution is started, the process stops at the beginning to the next step and a warning message is returned.

As the SQL COMMIT statement is called by the redistribute process, running the redistribute process under a Type-2 connection is not supported.

Chapter 18. Storage management tool routines

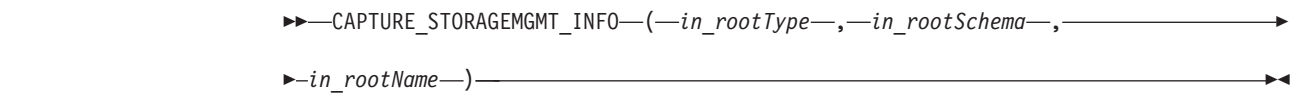
CAPTURE_STORAGEMGMT_INFO procedure - Retrieve storage-related information for a given root object

The CAPTURE_STORAGEMGMT_INFO procedure attempts to collect the storage-related information for the given root object, as well as the storage objects defined within its scope. All the storage objects are specified in the SYSTOOLS.STMG_OBJECT_TYPE table.

Table 234. STMG_OBJECT_TYPE table

Column name	Data type	Nullable	Description
OBJ_TYPE	INTEGER	N	Integer value corresponds to a type of storage object • 0 - Database • 1 - Database Partition Group • 2 - Table Space • 3 - Table Space Container • 4 - Table • 5 - Index
TYPE_NAME	VARCHAR	N	Descriptive name of the storage object type • STMG_DATABASE • STMG_DBPGROUP • STMG_TABLESPACE • STMG_CONTAINER • STMG_TABLE • STMG_INDEX

Syntax



The schema is SYSPROC.

Procedure parameters

in_rootType
An input argument of type SMALLINT. The valid option types are:

- 0 - Database
- 1 - Database Partition Group
- 2 - Table Space
- 4 - Table
- 5 - Index

The input argument cannot be null. If a null value is specified, an SQL0443 error with SQLSTATE 38553, and token DBA7617 is returned.

in_rootSchema
An input argument of type VARCHAR (128) that specifies the schema name of the storage snapshot root object.

in_rootName
An input argument of type VARCHAR (128) that specifies the name of the root object. The input argument cannot be null. If a null value is specified, an SQL0443 error with SQLSTATE 38553, and token DBA7617 is returned.

Authorization

- EXECUTE privilege on the CAPTURE_STORAGEMGMT_INFO procedure.
- EXECUTE privilege on the SYSPROC.DB_PARTITIONS, SYSPROC.SNAP_GET_CONTAINER, SYSPROC.SNAPSHOT_CNTRFS table functions.
- SELECT privilege on SYSCAT.TABLES, SYSCAT.TABLESPACES, SYSCAT.NODEGROUPDEF, SYSCAT.DATABASEPARTITIONS, SYSCAT.DATAPARTITIONEXPRESSION, SYSCAT.INDEXES, and SYSCAT.COLUMNS.

Usage Notes:

1. The following stored procedure should be used to create storage management tables:

   create_storagemgmt_tables(TABLESPACE_NAME) where 'TABLESPACE' is the name of the table space, on which storage management tables would be created.

   (In case of a problem with the existing storage management tables, it can be dropped using the below stored procedure and can be re-created using the above stored procedure.

   drop_storagemgmt_tables(0 or 1) where '0' or '1 'indicates 'stop' or 'proceed' on encountering an error respectively.)
2. The following command should be used to run statistics for the storage object for which details have to be obtained:

   RUNSTATS ON TABLE (TABLESCHEMA.TABLENAME) ON KEY COLUMNS AND INDEXES ALL
3. The following command should be used to populate the storage management tables:

   Run 'capture_storagemgnt_info()' stored procedure to populate the storage management tables. Sometimes it might be necessary to run the CAPTURE_STORAGEMGMT_INFO procedure twice. The first time you run it, use the CAPTURE_STORAGEMGMT_INFO procedure to populate the storage tables with table space details. For example:

   ```
   db2 "call capture_storagemgmt_info(0,(SCHEMA_NAME),(DATABASE_NAME))"
   ```

   The second time, use the CAPTURE_STORAGEMGMT_INFO procedure to add details about the storage of the actual object to the storage table. For example, the following example adds details for an object of type index (the *in_rootType* argument is set to 5):

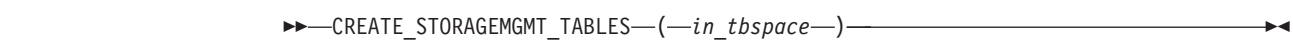
   ```
   db2 "call
   capture_storagemgmt_info(5,(SCHEMA_NAME),(SCHEMA_NAME.INDEX_NAME))"
   ```
4. Run the select query on the required storage management table to see the details of the storage object, for example: as follows in case of INDEX object :

   ```
   db2 "SELECT * FROM SYSTOOLS.STMG_INDEX"
   ```

CREATE_STORAGEMGMT_TABLES procedure - Create storage management tables

The CREATE_STORAGEMGMT_TABLES procedure creates all storage management tables under a fixed "DB2TOOLS" schema, in the table space specified by input.

Syntax



The schema is SYSPROC.

Procedure parameters

in_tbspace
An input argument of type VARCHAR(128) that specifies the table space name. The input argument cannot be null. If a null value is specified, an SQL0443 error with SQLSTATE 38553, and token DBA7617 is returned.

Authorization

EXECUTE privilege on the CREATE_STORAGEMGMT_TABLES procedure.

You must also have CREATETAB privilege on the database and USE privilege on the table space, and one of:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name DB2TOOLS does not exist.
- CREATEIN privilege on the schema, if the schema name of the table exists.
- DBADM authority

Usage notes

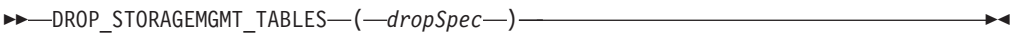
The following tables are created in the DB2TOOLS schema:

- STMG_CONTAINER
- STMG_CURR_THRESHOLD
- STMG_DATABASE
- STMG_DBPARTITION
- STMG_DBPGROUP
- STMG_HIST_THRESHOLD
- STMG_INDEX
- STMG_OBJECT
- STMG_OBJECT_TYPE
- STMG_ROOT_OBJECT
- STMG_TABLE
- STMG_TABLESPACE
- STMG_TBPARTITION
- STMG_THRESHOLD_REGISTRY

DROP_STORAGEMGMT_TABLES procedure - Drop all storage management tables

The DROP_STORAGEMGMT_TABLES procedure attempts to drop all storage management tables.

Syntax



The schema is SYSPROC.

Procedure parameters

dropSpec
An input argument of type SMALLINT. When *dropSpec* is set to 0, the process stops when any error is encountered; when *dropSpec* is set to 1, the process continues, ignoring any error it encounters. The input argument cannot be null. If a null value is specified, an SQL0443 error with SQLSTATE 38553, and token DBA7617 is returned.

Authorization

EXECUTE privilege on the DROP_STORAGEMGMT_TABLES procedure.

The user ID that establishes the database connection must either be the definer of the storage management tables as recorded in the DEFINER column of SYSCAT.TABLES, or have at least one of the following privileges:

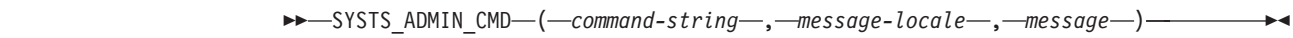
- DBADM authority
- DROPIN privilege on the schema for these tables
- CONTROL privilege on these tables

Chapter 19. Text Search routines

SYSTS_ADMIN_CMD stored procedure - Run text search administration commands

The SYSTS_ADMIN_CMD procedure is used by applications to run text search administrative commands using the SQL CALL statement.

Syntax



The schema is SYSPROC.

Procedure parameter

command-string
An input argument of type VARCHAR (32K) that specifies a single text search index administration command that is to be executed. The command syntax is the same as the DB2 Text Search command with the exception of the connection options. Connection options are not supported through this procedure. Commands that are run through this procedure use the current connection.

message-locale
An input argument of type VARCHAR (33) that specifies the desired language for any error message text returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR (32K) that specifies a warning or informational message for an operation that is considered successful.

Authorization

EXECUTE privilege on the SYSTS_ADMIN_CMD procedure.

The procedure currently supports the following DB2 Text Search commands:

- ALTER INDEX
- CLEAR COMMAND LOCKS
- CLEAR EVENTS
- CREATE INDEX
- DISABLE DATABASE
- DROP INDEX
- ENABLE DATABASE
- UPDATE INDEX

Example

Update text search index MYTEXTINDEX in schema DB2TS and return any error messages in English.

```
CALL SYSPROC.SYSTS_ADMIN_CMD
 ('UPDATE INDEX DB2TS.MYTEXTINDEX FOR TEXT','en_US', ?)";
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : CIE00001 Operation completed successfully.

Return Status = 0
```

Usage notes

- If the execution of the command is not successful, SQLCODE -20427 and SQLSTATE 38H14 is returned with the text search specific error message. For example, if index MYTEXTINDEX already exists and the following statement is issued:

  ```
  CALL SYSPROC.SYSTS_ADMIN_CMD ('CREATE INDEX MYTEXTINDEX FOR TEXT
    ON DB2TS.TEXTBOOKS (STORY)', 'en_US', ?)
  ```

  the index creation will fail with the following error message.

  ```
  SQL20427N An error occurred during a text search administration
  procedure or  command. The error message is "CIE00201 Text search
  index "DB2TS  "."MYTEXTINDEX" already exists. ". SQLSTATE=38H14
  ```

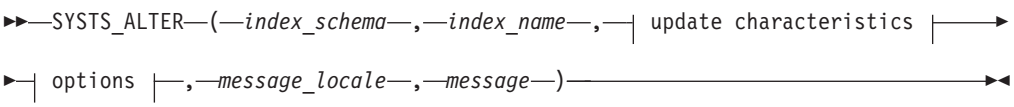
- If an SQLCODE is returned by the procedure, the message might be truncated. Full message information can be found in the db2diag log files.

SYSTS_ALTER procedure - Change the update characteristics of an index

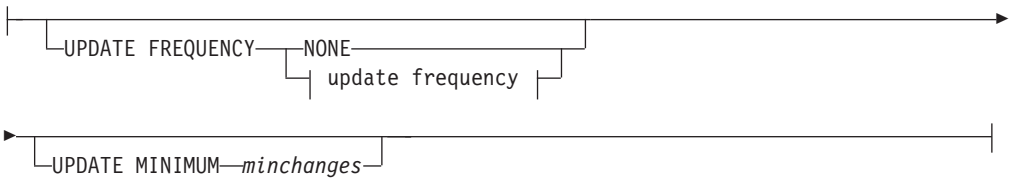
This procedure changes the update characteristics of an index.

The procedure issues an ALTER INDEX text search administration command on the database server.

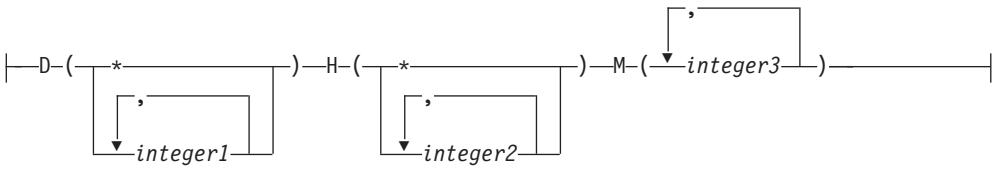
Syntax



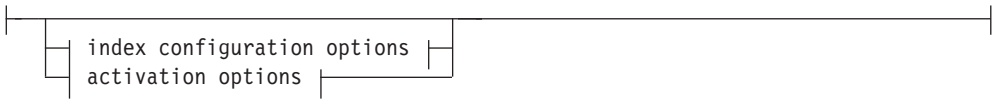
update characteristics:



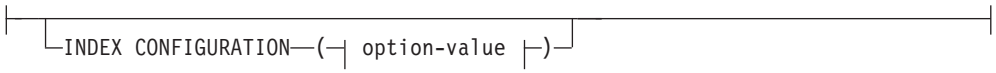
update frequency:



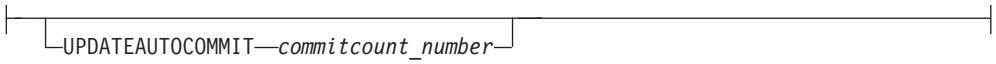
options:



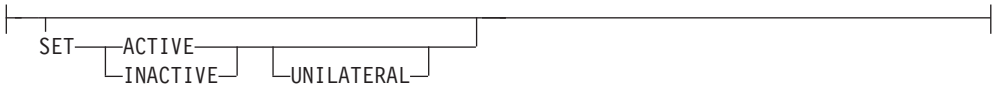
index configuration options:



option-value:



activation options:



The schema is SYSPROC.

Procedure parameters

index_schema
An input argument of type VARCHAR(128) that specifies the schema of the text search index. The *index_schema* must follow the naming restriction for DB2 schema names. If the argument is null or an empty string, the value of CURRENT SCHEMA is used. The *index_schema* is case-sensitive.

index_name
An input argument of type VARCHAR(128) that specifies the name of the index. Together with *index_schema*, it uniquely identifies a text search index in a database. The *index_name* is case-sensitive.

update characteristics
An input argument of type VARCHAR(32K) that specifies the alter options. The alter options allowed are as follows:

UPDATE FREQUENCY
Specifies the frequency with which index updates are made. The index will be updated, if the number of changes is at least the value set for UPDATE MINIMUM. The update frequency NONE indicates that no further index

updates will be made. This can be useful for a text column in a table with data that will not change. It is also useful when the user intends to manually update the index (using the UPDATE INDEX command). Automatic updates can only be done if the START FOR TEXT command has been run and the DB2 Text Search instance services are running.

The default frequency value is taken from the view SYSIBMTS.TSDEFAULTS, where DEFAULTNAME='UPDATEFREQUENCY'.

NONE
No automatic updates will be applied to the text index. Any further index update will have to be started manually.

D The day(s) of the week when the index is updated.

* Every day of the week.

integer1
Specific days of the week, from Sunday to Saturday: 0 to 6

H The hour(s) of the specified day(s) when the index is updated.

* Every hour of the day.

integer2
Specific hours of the day, from midnight to 11 pm: 0 to 23

M The minute(s) of the specified hour(s) when the index is updated.

integer3
Specified as top of the hour (0), or in multiples of 5 minute increments after the hour: 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 or 55

If you do not specify the UPDATE FREQUENCY option, the frequency settings are left unchanged.

UPDATE MINIMUM *minchanges*
Specifies the minimum number of changes to text documents that must occur before the index is incrementally updated. Multiple changes to the same text document are treated as separate changes. If you do not specify the UPDATE MINIMUM option, the setting is left unchanged.

INDEX CONFIGURATION (*option-value***)**
Starting with Version 9.7 Fix Pack 3 and later fix packs, this is an optional input argument of type VARCHAR(32K) that allows altering text index configuration settings. The following option is supported:

Table 235. Specifications for option-value

Option	Value	Data type	Description
UPDATEAUTOCOMMIT	*numCommit*	integer	Specifies the number of index updates after which a commit is executed to preserve the previous work automatically for either initial or incremental updates. • For initial updates, the index update will process batches of documents from a base table after the trigger to capture data updates is activated. After the amount of documents updated reaches the `COMMITCOUNT` number, the server will do an interim commit. Log entries generated by unprocessed documents will be removed from staging table. Using the `UPDATEAUTOCOMMIT` option for an initial text index update will lead to a significant increase of execution time. • For incremental updates, log entries which have been processed will be removed correspondingly from staging table with each interim commit. `COMMITCOUNT` counts the number of documents updated, not the number of staging table entries.

activation options
Starting with Version 9.7 Fix Pack 3 and later fix packs, this input argument of type integer sets the status of a text index.

ACTIVE
Sets the text index status to active

INACTIVE
Sets the text index status to inactive

UNILATERAL
Specifies a unilateral change which only affects the status of DB2 Text Search indexes. If this argument is specified, only the status of a DB2 Text Search index is changed to active or inactive. Without the `UNILATERAL` argument, the activation status of the DB2 Text Search and DB2 Net Search Extender indexes is jointly switched so that only one of the text indexes is active.

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table on which the text index is defined
- DBADM authority

Examples

Example 1: In the following example, the update characteristics of a text search index are being altered. This index was originally created with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. By using 'UPDATE FREQUENCY NONE', the intention is to make no further updates to the text search index as possibly no changes are expected for the associated table column. Any error messages are requested to be returned in English. When the procedure succeeds, the output parameter message indicative of the successful operation is returned to the caller.

```
CALL SYSPROC.SYSTS_ALTER('db2ts', 'myTextIndex',
 'UPDATE FREQUENCY NONE', 'en_US', ?)
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.

Return Status = 0
```

Example 2: In the following example, the SYSTS_ALTER stored procedure is called to alter the update-characteristics for a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. The intention is to ensure that updates to the index occur every hour on the hour. However, this index does not exist and results in an error.

```
CALL SYSPROC.SYSTS_ALTER('db2ts', 'myTextIndex',
 'update frequency D(*) H(*) M(0)', 'en_US', ?)
```

The following is an example of output from this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message is "CIE00316 Text search
index "db2ts"."myTextIndex" does not exist. ". SQLSTATE 38H14
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT for the connection.
- Multiple procedures or commands cannot be run concurrently on a text search index if they might conflict. Some of the conflicting procedures and commands are:
  - SYSTS_ALTER procedure or ALTER INDEX db2ts command
  - SYSTS_CLEAR_EVENTS procedure or CLEAR EVENTS FOR INDEX db2ts command
  - SYSTS_DISABLE procedure or DISABLE DATABASE FOR TEXT db2ts command
  - SYSTS_DROP procedure or DROP INDEX db2ts command
  - STOP FOR TEXT db2ts command
  - SYSTS_UPDATE procedure or UPDATE INDEX db2ts command

If there is a conflict, the procedure returns an SQLCODE -20426 and SQLSTATE 38H13.

- When this procedure is run,
  - the content of the DB2 Text Search view SYSIBMTS.TSLOCKS is updated.
  - the index entry in the Text Search Index Data file is updated. The file contains a persistent representation of update schedules (also empty ones) for each index in the instance.
- The result of activating indexes depends on the original index status. The table below describes the results.

Table 236. Status changes without invalid index:

Initial DB2 Text Search or Net Search Extender Status	Request Active	Request Active Unilateral	Request Inactive	Request Inactive Unilateral
Active / Inactive	No change	No change	Inactive / Active	Inactive / Inactive
Inactive / Active	Active / Inactive	Error	No change	No change
Inactive / Inactive	Active / Inactive	Active / Inactive	Inactive / Active	No change

`SQL20427N` and `CIE0379E` error messages are returned for active index conflicts.

SYSTS_CLEAR_COMMANDLOCKS procedure - Remove command locks for text search indexes

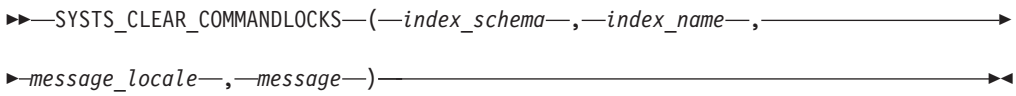
This procedure removes all command locks for a specific text search index or for all text search indexes in the database.

A command lock is created at the beginning of a text search index command, and is destroyed when the command has completed. It prevents undesirable conflict between different commands.

A cleanup is done automatically of all locks associated with processes that are no longer alive. This is done to make a text search index accessible to a new search request. Use of this procedure is required in the rare case that locks remain in place due to an unexpected system behavior, and need to be cleaned up explicitly.

This procedure issues the CLEAR COMMAND LOCKS text search administration command on the database server.

Syntax



The schema is SYSPROC.

Procedure parameters

index_schema
An input argument of type VARCHAR(128) that specifies the schema of the text index. The *index_schema* must follow the naming restriction for DB2

schema names. If the argument is null or an empty string, the value of CURRENT SCHEMA is used. The *index_schema* is case-sensitive.

index_name
An input argument of type VARCHAR(128) that specifies the name of the index. Together with *index_schema*, it uniquely identifies a text search index in a database. If the argument is null or an empty string, the procedure deletes command locks for all text search indexes in the database. The *index_name* is case-sensitive.

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The *username* for the database connection must have DBADM authority if an index name is not specified. For clearing a command lock on a specific index, the *username* for the database connection must have CONTROL privilege on the table for which the text search index was created.

Examples

Example 1: In the following example, SYSTS_CLEAR_COMMANDLOCKS is issued for a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. Error messages are requested to be returned in English. When the procedure succeeds, the output parameter message indicative of the successful operation is returned to the caller.

```
CALL SYSPROC.SYSTS_CLEAR_COMMANDLOCKS('db2ts', 'myTextIndex', 'en_US', ?)
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.

Return Status = 0
```

Example 2: In the following example, SYSTS_CLEAR_COMMANDLOCKS is called to clear the command locks for a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. This index does not exist and the procedure returns an error message.

```
CALL SYSPROC.SYSTS_CLEAR_COMMANDLOCKS('db2ts', 'myTextIndex', 'en_US', ?)
```

The following is an example of output from this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message is "CIE00316 Text search
index "db2ts"."myTextIndex" does not exist. ". SQLSTATE 38H14
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending

on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT.

- The process and thread information in the view SYSIBMTS.TSLOCKS can be used to check if the thread or process that holds the lock still exists. The locks for existing processes belonging to running text search administration procedure or command (for example, SYSTS_UPDATE or UPDATE INDEX) should not be cleared.
- You would invoke this procedure because the process owning the command lock is dead. In this case, the command (represented by the lock) may not have completed, and the index may not be operational. You need to take appropriate action. For example, the process executing the DROP INDEX command dies suddenly. It has deleted some index data, but not all the catalog and collection information. The command lock is left intact. After clearing the DROP INDEX command lock, you may wish to re-execute the SYSTS_DROP procedure. In another example, the process executing the SYSTS_CREATE procedure dies suddenly. It has created some index catalog and collection information, but not all. The command lock is left intact. After clearing the command lock, you can execute the SYSTS_DROP and SYSTS_CREATE procedures.
- When this procedure is run, the content of the DB2 Text Search view SYSIBMTS.TSLOCKS is updated.

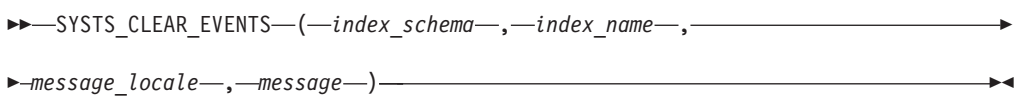
SYSTS_CLEAR_EVENTS procedure - Delete indexing events from an index's event table

This procedure deletes indexing events from an index's event table used for administration.

The name of the event table can be found in the view SYSIBMTS.TSINDEXES in column EVENTVIEWNAME. Every index update operation that processes at least one document produces informational and, in some cases, error entries in the event table. For automatic updates, the event table has to be regularly inspected. Document specific errors must be corrected by changing the document content. After correcting the errors, the events can be cleared (and should be, in order not to consume too much space).

The procedure issues a CLEAR EVENTS FOR INDEX text search administration command on the database server.

Syntax



The schema is SYSPROC.

Procedure parameters

index_schema
An input argument of type VARCHAR(128) that specifies the schema of the text search index. The *index_schema* must follow the naming restriction for DB2 schema names. If the argument is null or an empty string, the value of CURRENT SCHEMA is used. The *index_schema* is case-sensitive.

index_name
An input argument of type VARCHAR(128) that specifies the name of the index. Together with *index_schema*, it uniquely identifies a text search index in a database. The *index_name* is case-sensitive.

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table on which the index is defined
- DBADM authority

Examples

Example 1: In the following example, SYSTS_CLEAR_EVENTS is being called for a text search index that was created with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. Any error messages are requested to be returned in English. When the procedure succeeds, the output parameter message indicative of the successful operation is returned to the caller.

```
CALL SYSPROC.SYSTS_CLEAR_EVENTS('db2ts', 'myTextIndex', 'en_US', ?)
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.

Return Status = 0
```

Example 2: In the following example, SYSTS_CLEAR_EVENTS is called to clear the event table entries for a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. This index does not exist and results in an error.

```
CALL SYSPROC.SYSTS_CLEAR_EVENTS('db2ts', 'myTextIndex', 'en_US', ?)
```

The following is an example of output from this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message is "CIE00316 Text search
index "db2ts"."myTextIndex" does not exist. ". SQLSTATE 38H14
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT.

- Multiple procedures or commands cannot be run concurrently on a text search index if they might conflict. Some of the conflicting procedures and commands are:
  - SYSTS_ALTER procedure or ALTER INDEX db2ts command
  - SYSTS_DISABLE procedure or DISABLE DATABASE FOR TEXT db2ts command
  - SYSTS_DROP procedure or DROP INDEX db2ts command
  - STOP FOR TEXT db2ts command
  - SYSTS_UPDATE procedure or UPDATE INDEX db2ts command

  If there is a conflict, the procedure returns an SQLCODE -20426 and SQLSTATE 38H13.
- When regular updates are scheduled (see UPDATE FREQUENCY options in SYSTS_CREATE or SYSTS_ALTER procedures), the event table should be checked regularly.
- To clean up the DB2 Text Search event table for a text search index, use the SYSTS_CLEAR_EVENTS procedure or CLEAR EVENTS FOR INDEX db2ts command after you have checked the reason for the event and removed the source of the error.
- Ensure that changes have been made to all rows referenced in the event table. By changing the rows in the user table, you ensure that when you run the SYSTS_UPDATE procedure or UPDATE INDEX db2ts command again, an attempt is made to index the erroneous documents again.
- When this command is issued, the event table is cleared.

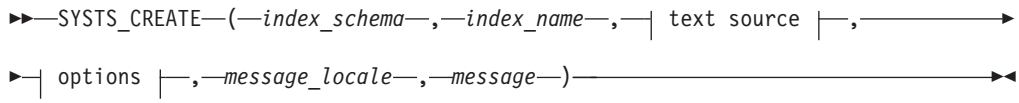
SYSTS_CREATE procedure - Create a text search index on a column

This procedure creates a text search index for a text column which allows the column data to be searched using text search functions.

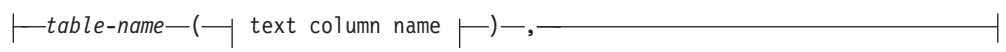
Once the text search index is created, the column can be searched using text search functions in queries. The index will not contain any data until the text search UPDATE INDEX command or SYSTS_UPDATE procedure is explicitly executed by the user, or implicitly executed by the text search instance level services, according to the defined update frequency for the index.

The procedure issues a CREATE INDEX text search administration command on the database server.

Syntax

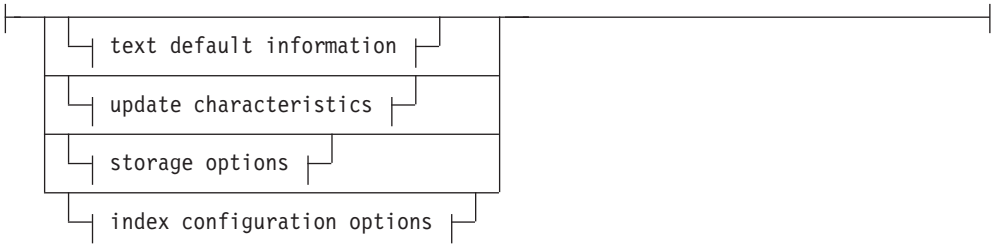


text source:

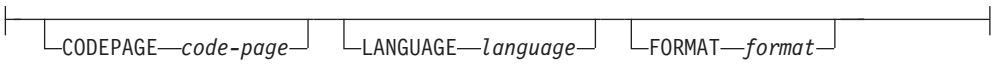


text column name:

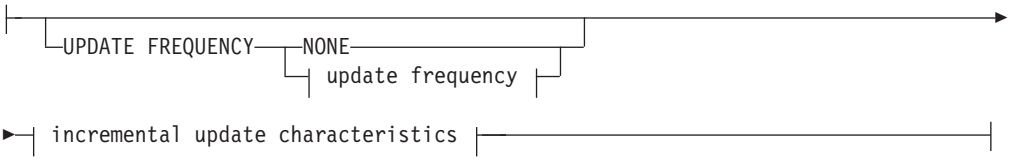
options:



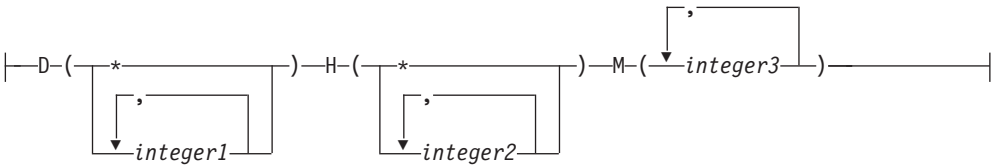
text default information:



update characteristics:

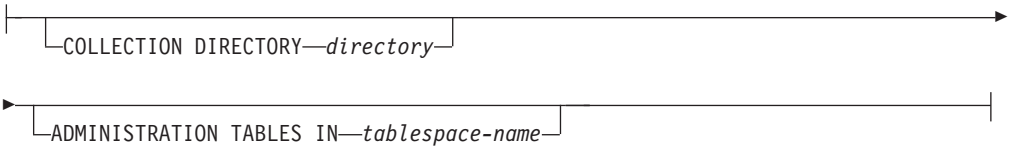


update frequency:

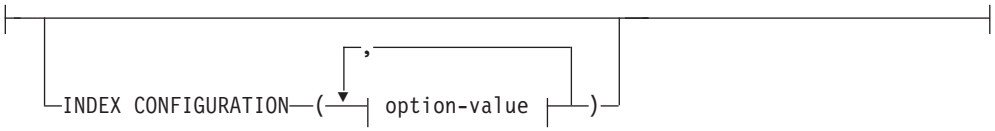


incremental update characteristics:

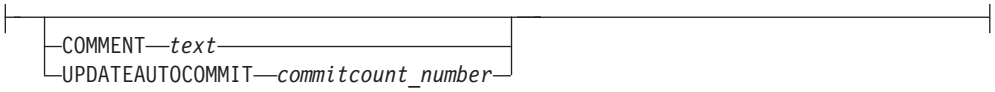
storage options:



index configuration options:



option-value:



The schema is SYSPROC.

Procedure parameters

index_schema
An input argument of type VARCHAR(128) that specifies the schema of the text search index. The *index_schema* must follow the naming restriction for DB2 schema names. If the argument is null or an empty string, the value of CURRENT SCHEMA is used. The *index_schema* is case-sensitive.

index_name
An input argument of type VARCHAR(128) that specifies the name of the index. Together with *index_schema*, it uniquely identifies a text search index in a database. The *index_name* is case-sensitive.

text source
An input argument of type VARCHAR(1024) that specifies the name of the column to be indexed. The options are:

table-name
The table name containing the text column. Text search indexes cannot be created on the following tables:

- range-partitioned tables
- federated tables
- materialized query tables
- views

The *table-name* is case-sensitive.

text column name
The column name of the column to be indexed.

column-name
The column must be of one of the following data types: CHAR, VARCHAR, LONG VARCHAR, CLOB, DBCLOB, BLOB, GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, or XML. If the data type of the column is not one of these, use a transformation function specified with *function-schema.function-name* to convert the column type to one of the valid types. Refer to the *function-name (column-name)* for syntax and details. Alternatively, you can specify a user-defined external function that accesses the text documents to be indexed. Only a single text search index can be created for a column. The *column-name* is case-sensitive.

function-name (column-name)
Specifies the schema qualified name, that conforms to DB2 naming conventions, of an external scalar function that accesses text documents in a column that is not of a supported type for text searching. Performs a data type conversion of that value, and returns the value as one of the supported data types for text searching. Its task is to perform a column type conversion. This function must take only one parameter and return only one value. The *function-name (column-name)* is case-sensitive.

options
An input argument of type VARCHAR(32K) that specifies the options to be used. If no options are needed, the argument can be null or an empty string. The available options are:

CODEPAGE *code-page*
Specifies the DB2 code page (CODEPAGE) to be used when indexing text documents. The default value is specified by the value in the view SYSIBMTS.TSDEFAULTS, where DEFAULTNAME='CODEPAGE' (which happens to be the database code page). This argument only applies to binary data types, i.e., the column type or return type from a transformation function must be BLOB or character-type FOR BIT DATA.

LANGUAGE *language*
Specifies the language to be used by DB2 Text Search for language specific processing of a document during indexing. If you do not specify a locale, the database territory will be used to determine the default setting for LANGUAGE. If you would like to have your documents automatically scanned to determine the locale, specify locale as AUTO.

FORMAT *format*
Specifies the format of text documents in the column. The supported formats include: TEXT, XML, and HTML. DB2 Text Search needs this information when indexing documents. If the format is not specified, the default value is used. The default value is in the view SYSIBMTS.TSDEFAULTS, where DEFAULTNAME='FORMAT'. For columns of data type XML, the default format 'XML' is used, regardless of the value of DEFAULTNAME.

UPDATE FREQUENCY
Specifies the frequency with which index updates are made. The index will be updated, if the number of changes is at least the value set for UPDATE MINIMUM. The update frequency NONE indicates that no further index updates will be made. This can be useful for a text column in a table with data that will not change. It is also useful when the user intends to manually update the index (using the UPDATE INDEX command). Automatic updates can only be done if the START FOR TEXT command has been run and the DB2 Text Search instance services are running.

The default frequency value is taken from the view SYSIBMTS.TSDEFAULTS, where DEFAULTNAME='UPDATEFREQUENCY'.

NONE
No further index updates are made. The update has to be started manually.

D The day(s) of the week when the index is updated.

* Every day of the week.

integer1
Specific days of the week, from Sunday to Saturday: 0 to 6

H The hour(s) of the specified day(s) when the index is updated.

***** Every hour of the day.

integer2
Specific hours of the day, from midnight to 11 pm: 0 to 23

M The minute(s) of the specified hour(s) when the index is updated.

integer3
Specified as top of the hour (0), or in multiples of 5 minute increments after the hour: 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 or 55

UPDATE MINIMUM *minchanges*
Specifies the minimum number of changes to text documents before the index is updated incrementally at the time specified in UPDATE FREQUENCY. Positive integer values only are allowed. The default value is taken from the view SYSIBMTS.TSDEFAULTS, where DEFAULTNAME='UPDATEMINIMUM'.

Note: This value is ignored during an UPDATE INDEX command (unless the USING UPDATE MINIMUM option is used there). A small value increases consistency between the table column and the text search index. However, it also causes higher performance overhead.

COLLECTION DIRECTORY *directory*
The directory in which the text search index is stored. By default the collection data will be located in `DBPATH/db2collections` where the value for `DBPATH` is the file path used for creating the database. You must specify the absolute path. The maximum length of the absolute path name is 215 characters. Different indexes are organized in subdirectories named *index identifier* under `COLLECTION DIRECTORY` where index identifier is a system generated identifier.

ADMINISTRATION TABLES IN *tablespace-name*
Specifies the name of an existing regular table space for the administration tables created for the index. If not specified, the table space of the base table for which the index is being created is used.

INDEX CONFIGURATION (*option-value***)**
Specifies additional index related values as option value string pairs. The following values are supported:

Table 237. Specifications for option-value

Option	Value	Data type	Description
COMMENT	*text*	String value less than 512 bytes	Adds a string comment value to the `REMARKS` column in the DB2 Text Search catalog view `TSINDEXES`. It also adds the string comment value as the description of the collection.

Table 237. Specifications for option-value (continued)

Option	Value	Data type	Description
UPDATEAUTOCOMMIT	*non-zero integer*	Integer	Starting with Version 9.7 Fix Pack 3 and later fix packs, it specifies the number of index updates after which a commit is executed to preserve the previous work automatically for either initial or incremental updates. • For initial updates, the index update will process batches of documents from a base table after the trigger to capture data updates is activated. After the amount of documents updated reaches the `COMMITCOUNT` number, the server will do an interim commit. Log entries generated by unprocessed documents will be removed from staging table. Using the `UPDATEAUTOCOMMIT` option for an initial text index update will lead to a significant increase of execution time. • For incremental updates, log entries which have been processed will be removed correspondingly from the staging table with each interim commit. `COMMITCOUNT` counts the number of documents updated, not the number of staging table entries.

Remember: Non-numeric values must be enclosed in single quotes. A single quote character within a string value must be represented by two consecutive single quotes.

Example:

```
INDEX CONFIGURATION (COMMENT 'Index on User''s Guide column')
```

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

One of:
- CONTROL privilege on the table on which the index is defined
- INDEX privilege on the table on which the index is defined

  and one of the following:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist

  - CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema
- DBADM authority

Examples

Example 1: In the following example, a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'` is created using the SYSTS_CREATE procedure. The option `'UPDATE MINIMUM 10'` specifies that at least 10 changes should be made to the text documents associated with the index before an incremental update of the index should be performed. Any error messages are requested to be returned in English. When the underlying text search command runs successfully, the output parameter `message` is set to indicate the status of the command execution.

```
CALL SYSPROC.SYSTS_CREATE('db2ts', 'myTextIndex',
 'myUserSchema.myBaseTable (myTextColumn)', 'UPDATE MINIMUM 10',
 'en_US', ?)
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.
Return Status = 0
```

Example 2: In the following example, SYSTS_CREATE is called to create a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. No options are specified. In this example, the index already exists which results in an error message being returned to the caller.

```
CALL SYSPROC.SYSTS_CREATE('db2ts', 'myTextIndex',
 'myUserSchema.myBaseTable (myTextColumn)', '', 'en_US', ?)
```

The following is an example of output from this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message is "CIE00201 Text search
index "db2ts"."myTextIndex" already exists. ".
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT for the connection.
- Multiple procedures or commands cannot be executed concurrently on a text search index if they might conflict. Some of the conflicting procedures and commands are:
  - SYSTS_ALTER procedure or ALTER INDEX db2ts command
  - SYSTS_CLEAR_EVENTS procedure or CLEAR EVENTS FOR INDEX db2ts command
  - SYSTS_DISABLE procedure or DISABLE DATABASE FOR TEXT db2ts command
  - STOP FOR TEXT db2ts command
  - SYSTS_UPDATE procedure or UPDATE INDEX db2ts command

If there is a conflict, the procedure returns an SQLCODE -20426 and SQLSTATE 38H13.

- With the successful execution of the CREATE INDEX command:
  - DB2 Text Search server data is updated. A collection of name *instance_database-name_index-identifier_number* is created, as in the following example:

    ```
    tigertail_MYTSDB_TS250517_0000
    ```

    The collection name can be retrieved from the SYSIBMTS.TSCOLLECTIONNAMES view (column COLLECTIONNAME).
  - DB2 Text Search catalog information is updated. An index staging table is created in the specified table space with appropriate DB2 indexes. In addition, an index event table is created in the specified table space.
  - When DB2 Text Search coexists with DB2 Net Search Extender and an active Net Search Extender index already exists for the table column, the new text index is set to inactive.
  - The newly created text search index is not automatically populated. The SYSTS_UPDATE procedure or UPDATE INDEX command must be executed either manually or automatically (as a result of an update schedule having been defined for the index through the specification of the UPDATE FREQUENCY option) for the text search index to be populated.
  - The Text Search index data file on the DB2 database server is updated. Scheduled update information is recorded for each index in the instance.

Usage restrictions:

- A primary key must be defined for the table. In DB2 Text Search, a multicolumn DB2 primary key can be used without type limitations. The number of primary key columns is limited to 2 columns less than the number of primary key columns allowed by DB2.
- The total length of all primary key columns for a table with DB2 Text Search indexes is limited to 15 bytes less than the maximum total primary key length allowed by DB2. Refer to the DB2 restrictions of DB2 CREATE INDEX statement.

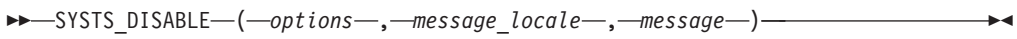
SYSTS_DISABLE procedure - Disable current database for text search

The procedure disables DB2 Text Search for the current database.

Once the Text Search feature has been disabled, text search indexes and commands are no longer available for use with the database.

The procedure issues a DISABLE DATABASE FOR TEXT text search administration command on the database server.

Syntax



The schema is SYSPROC.

Procedure parameters

options
An input argument of type VARCHAR(128) that specifies the options to be

used when disabling the database. The argument can be set to FORCE. When this value is specified, all indexes are dropped and the Text Search feature is disabled by force. No text search indexes are preserved and no error message or warning is returned. If the argument is null or an empty string, an attempt is made to disable the Text Search feature for the database.

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The privileges held by the authorization ID of the statement must include DBADM authority.

Examples

Example 1: In the following example, Text Search is disabled for a database using the SYSTS_DISABLE procedure. The FORCE option is specified to ensure that the feature is disabled even if text search indexes still exist on tables in the database. Error messages are specified requested to be returned in English. The *message* output parameter is set to an informational message string.

```
CALL SYSPROC.SYSTS_DISABLE('FORCE', 'en_US', ?)
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.

Return Status = 0
```

Example 2: In the following example, Text Search is disabled for a database with existing text search indexes using the SYSTS_DISABLE procedure without specifying the FORCE option. This results in an error message to the caller. It is preferable to drop all existing text search indexes prior to disabling the Text Search feature or alternatively to specify the FORCE option for the *options* input parameter value.

```
CALL SYSPROC.SYSTS_DISABLE('', 'en_US', ?)
```

The following is an example of output from this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message is "CIE00326 Text search
index active in specified or default database. ". SQLSTATE 38H14
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT.

- Multiple procedures or commands cannot be executed concurrently on a text search index if they might conflict. Some of the conflicting procedures and commands are:
  - SYSTS_ALTER procedure or ALTER INDEX db2ts command
  - SYSTS_CLEAR_EVENTS procedure or CLEAR EVENTS FOR INDEX db2ts command
  - SYSTS_DISABLE procedure or DISABLE DATABASE FOR TEXT db2ts command
  - STOP FOR TEXT db2ts command
  - SYSTS_UPDATE procedure or UPDATE INDEX db2ts command

  If there is a conflict, the procedure returns an SQLCODE -20426 and SQLSTATE 38H13.
- When this procedure is run,
  - the DB2 Text Search catalog information is updated. The index log and event tables are dropped. Triggers on the user text table are deleted.
  - if the FORCE option is specified, all text index information is removed from the database and all associated collections are deleted. In addition, the text service is updated to remove any remaining update schedule information. See the "db2ts DROP INDEX command" or "SYSTS_DROP procedure" for reference.
- This procedure does not influence the DB2 Net Search Extender enablement status of the database. It deletes the DB2 Text Search catalog tables and views that are created by the SYSTS_ENABLE procedure or the ENABLE FOR TEXT command.
- Before dropping a DB2 database that has text search index definitions, run this procedure and make sure that the text indexes and collections have been removed successfully.
- If some indexes could not be deleted using the FORCE option, the collection names are written to the db2diag log files. If the text search index procedure SYSTS_DISABLE or the command DISABLE DATABASE FOR TEXT is not executed before the CLP command DROP DATABASE, the text search index services must also be cleaned up using the CLEANUP FOR TEXT command. See the SYSTS_DROP procedure or DROP INDEX command for more about dropping indexes, and the CLEANUP FOR TEXT command for information about text search collections and their relationship to text search indexes.

  Note: The user is discouraged from usage that results in orphaned collections, that is, collections that remain defined on the text search server but are not used by DB2. Here are some cases that may cause orphaned collections:
  - When a DROP DATABASE CLP command or DROP TABLE statement is executed without running the SYSTS_DISABLE procedure or a DISABLE DATABASE FOR TEXT command.
  - When the SYSTS_DISABLE procedure is run or a DISABLE DATABASE FOR TEXT command is executed using the FORCE option.
  - Some other error conditions. The CLEANUP FOR TEXT command can be used in some scenarios.

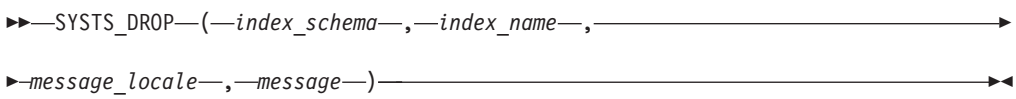
SYSTS_DROP procedure - Drop a text search index

This procedure drops an existing text search index associated with any table column.

After successful execution of this procedure, text search queries cannot be run on that column.

The procedure issues a DROP INDEX text search administration command on the database server.

Syntax



The schema is SYSPROC.

Procedure parameters

index_schema
An input argument of type VARCHAR(128) that specifies the schema of the text search index. The *index_schema* must follow the naming restriction for DB2 schema names. If the argument is null or an empty string, the value of CURRENT SCHEMA is used. The *index_schema* is case-sensitive.

index_name
An input argument of type VARCHAR(128) that specifies the name of the index. Together with *index_schema*, it uniquely identifies a text search index in a database. The *index_name* is case-sensitive.

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table on which the index is defined
- DBADM authority

Examples

Example 1: In the following example, the text search index that was created with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'` is being dropped. Any error messages are requested to be returned in English. When the procedure succeeds, the output parameter message indicative of the successful operation is returned to the caller.

```
CALL SYSPROC.SYSTS_DROP('db2ts', 'myTextIndex', 'en_US', ?)
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.

Return Status = 0
```

Example 2: In the following example,SYSTS_DROP is called to drop a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. This index does not exist and results in an error.

```
CALL SYSPROC.SYSTS_DROP('db2ts', 'myTextIndex', 'en_US', ?)
```

The following is an example of output from this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message is "CIE00316 Text search
index "db2ts"."myTextIndex" does not exist. ". SQLSTATE 38H14
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT.
- Multiple procedures or commands cannot be executed concurrently on a text search index if they might conflict. Some of the conflicting procedures and commands are:
  - SYSTS_ALTER procedure or ALTER INDEX db2ts command
  - SYSTS_CLEAR_EVENTS procedure or CLEAR EVENTS FOR INDEX db2ts command
  - SYSTS_DISABLE procedure or DISABLE DATABASE FOR TEXT db2ts command
  - STOP FOR TEXT db2ts command
  - SYSTS_UPDATE procedure or UPDATE INDEX db2ts command

  If there is a conflict, the procedure returns an SQLCODE -20426 and SQLSTATE 38H13.
- Dropping the user table in DB2 does not trigger dropping of indexes, they must be dropped manually before or after dropping the table.
- When this procedure is run,
  - the text search catalog information is updated. The index staging and event tables are dropped. Triggers on the user table are deleted.
  - the index entry in the Text Search Index Data file is deleted. The file contains a persistent representation of update schedules (also empty ones) for each index in the instance.
  - the collection associated with the text search index definition is removed.
- If, after dropping a text search index, you plan to create a new one on the same text column, you must first disconnect from and then reconnect to the database before creating the new text search index.

SYSTS_ENABLE procedure - Enable current database for text search

This procedure enables DB2 Text Search for the current database.

This procedure must be issued successfully before text search indexes on columns in tables within the database can be created.

This procedure issues the ENABLE DATABASE FOR TEXT text search administration command on the database server.

Syntax

The schema is SYSPROC.

Procedure parameters

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The user must have DBADM privilege to execute the ENABLE DATABASE command.

Examples

Example 1: Enable the database for text search and return any error messages in English.

```
CALL SYSPROC.SYSTS_ENABLE('en_US', ?)
```

The following is an example of output for this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.

Return Status = 0
```

Example 2: In the following example, SYSTS_ENABLE is called on a database that is already enabled for text search. This results in an error message to the caller.

```
CALL SYSPROC.SYSTS_ENABLE('en_US', ?)
```

The following is an example of output for this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message from the text search
product is "CIE00322 Specified or default database already
enabled for text. ". SQLSTATE 38H14
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending

on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT.

- When this procedure is run,
  - this procedure creates database objects, such as text search administration catalog tables and views, in the schema SYSIBMTS. These objects are placed in the default table space of the database (IBMDEFAULTGROUP).
  - the established database defaults for text search index are available in view SYSIBMTS.TSDEFAULTS.
  - and when the command has successfully completed, the text search catalog tables and views are created and are available.

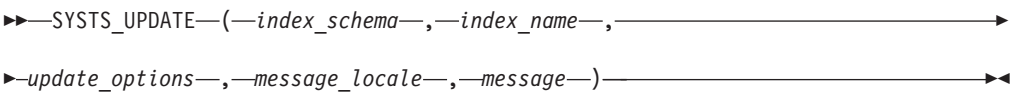
SYSTS_UPDATE procedure - Update the text search index

This procedure updates the text search index to reflect the current contents of the text column with which the index is associated.

While the update is being performed, a search is possible. Until completion of the update, the search operates on a partially updated index.

The procedure issues an UPDATE INDEX text search administration command on the database server.

Syntax



The schema is SYSPROC.

Procedure parameters

index_schema
An input argument of type VARCHAR(128) that specifies the schema of the text search index. The *index_schema* must follow the naming restriction for DB2 schema names. If the argument is null or an empty string, the value of CURRENT SCHEMA is used. The *index_schema* is case-sensitive.

index_name
An input argument of type VARCHAR(128) that specifies the name of the index. Together with *index_schema*, it uniquely identifies a text search index in a database. The *index_name* is case-sensitive.

update_options
An input argument of type VARCHAR(32K) that specifies update options. The possible values are:

- USING UPDATE MINIMUM: this setting respects the UPDATE MINIMUM settings from the CREATE INDEX text search administration command and the SYSTS_CREATE procedure.
- UPDATEAUTOCOMMIT: this setting overrides the commitcount defined for the text index for the duration of this update execution.
- NULL or an empty string (''): the update is unconditionally started when the procedure is called.

message_locale
An input argument of type VARCHAR(33) that specifies the locale to be used for any error message returned. If the argument is null or an empty string, or the message files for the specified locale are not available on the server, 'en_US' is used.

message
An output argument of type VARCHAR(32K) that specifies a warning or informational message for a successfully completed operation.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table on which the text index is defined
- DATAACCESS authority

Examples

Example 1: In the following example, the text search index that was created with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'` is being updated. A NULL value in the place of the *update_options* means that an update is unconditionally started when the stored procedure is called. Any error messages are requested to be returned in English. When the procedure succeeds, the output parameter message indicative of the successful operation is returned to the caller.

```
CALL SYSPROC.SYSTS_UPDATE('db2ts', 'myTextIndex', '', 'en_US', ?)
```

The following is an example of output from this query.

```
Value of output parameters
--------------------------
Parameter Name  : MESSAGE
Parameter Value : Operation completed successfully.

Return Status = 0
```

Example 2: In the following example, SYSTS_UPDATE is called to update a text search index with *index_schema* `'db2ts'` and *index_name* `'myTextIndex'`. This index does not exist and results in an error.

```
CALL SYSPROC.SYSTS_UPDATE('db2ts', 'myTextIndex', 'USING UPDATE MINIMUM',
'en_US', ?)
```

The following is an example of output from this query.

```
SQL20427N An error occurred during a text search administration
procedure or command. The error message is "CIE00316 Text search
index "db2ts"."myTextIndex" does not exist. ". SQLSTATE 38H14
```

Usage notes

- Text search administration procedures use an existing connection to the database. The current transaction might be committed or rolled back depending on the completion of the procedures. As such, you might want to commit all transaction changes to avoid any unexpected impact from such a commit or rollback. One way to achieve this is to turn on AUTOCOMMIT.
- Multiple procedures or commands cannot be run concurrently on a text search index if they might conflict. Some of the conflicting procedures and commands are:
  - SYSTS_ALTER procedure or ALTER INDEX db2ts command

- SYSTS_CLEAR_EVENTS procedure or CLEAR EVENTS FOR INDEX db2ts command
- SYSTS_DISABLE procedure or DISABLE DATABASE FOR TEXT db2ts command
- SYSTS_DROP procedure or DROP INDEX db2ts command
- STOP FOR TEXT db2ts command
- SYSTS_UPDATE procedure or UPDATE INDEX db2ts command

If there is a conflict, the procedure returns an SQLCODE -20426 and SQLSTATE 38H13.

- This procedure does not return until all index update processing is completed. The duration depends on the number of documents to be indexed and the number of documents already indexed. The collection name for the index can be retrieved from the SYSIBMTS.TSCOLLECTIONNAMES view (column COLLECTIONNAME).
- When there are individual document errors, the documents must be corrected. The primary keys of the erroneous documents can be looked up in the event table for the index. By changing the corresponding rows in the user table, the next call to SYSTS_UPDATE will reprocess these documents.
- When this procedure is run,
  - rows are inserted into the event table (including parser error information). Information is deleted from the index staging table in case of incremental updates. Before the first update, it creates triggers on the user table.
  - the collection is updated: new or changed documents are parsed and indexed and deleted documents are discarded from the index.

Chapter 20. Workload Management routines

WLM_CANCEL_ACTIVITY - Cancel an activity

This procedure cancels a given activity. If the cancel takes place, an error message will be returned to the application that submitted the activity that was cancelled.

Syntax

The schema is SYSPROC.

Procedure parameters

application_handle
An input argument of type BIGINT that specifies the application handle whose activity is to be cancelled. If the argument is null, no activity will be found and an SQL4702N with SQLSTATE 5U035 is returned.

uow_id
An input argument of type INTEGER that specifies the unit of work ID of the activity that is to be cancelled. If the argument is null, no activity will be found and an SQL4702N with SQLSTATE 5U035 is returned.

activity_id
An input argument of type INTEGER that specifies the activity ID which uniquely identifies the activity within the unit of work that is to be cancelled. If the argument is null, no activity will be found and an SQL4702N with SQLSTATE 5U035 is returned.

Authorization

EXECUTE privilege on the WLM_CANCEL_ACTIVITY procedure.

Example

An administrator can use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function to find the application handle, unit of work ID and activity ID of an activity. To cancel an activity with application handle 1, unit of work ID 2 and activity ID 3:

```
CALL WLM_CANCEL_ACTIVITY(1, 2, 3)
```

Usage notes

- If no activity can be found, an SQL4702N with SQLSTATE 5U035 is returned.
- If the activity cannot be cancelled because it not in the correct state (not initialized), an SQL4703N (reason code 1) with SQLSTATE 5U016 is returned.
- If the activity is successfully cancelled, an SQL4725N with SQLSTATE 57014 is returned to the cancelled application.
- If, at the time of the cancel, the coordinator is processing a request for a different activity or is idle, the activity is placed into CANCEL_PENDING state and will be cancelled when the coordinator processes the next request.

WLM_CAPTURE_ACTIVITY_IN_PROGRESS - Collect activity information for activities event monitor

The WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure gathers information about a specified activity and writes the information to the active activities event monitor.

When you apply this procedure to an activity with child activities, the procedure recursively generates a record for each child activity. This information is collected and sent when you call the procedure; the procedure does not wait until the parent activity completes execution. The record of the activity in the event monitor is marked as a partial record.

Syntax

The schema is SYSPROC.

Procedure parameters

If you do not specify all of the following parameters, no activity is found, and SQL4702N with SQLSTATE 5U035 is returned.

application_handle
An input argument of type BIGINT that specifies the handle of the application whose activity information is to be captured.

uow_id
An input argument of type INTEGER that specifies the unit of work ID of the activity whose information is to be captured.

activity_id
An input argument of type INTEGER that specifies the activity ID that uniquely identifies the activity within the unit of work whose information is to be captured.

Authorization

EXECUTE privilege on the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure.

Example

Assume that a user complains that stored procedure MYSCHEMA.MYSLOWSTP seems to be running more slowly than usual. The administrator wants to investigate the cause of the slowdown. Investigating while the stored procedure is running is not practical, so the administrator decides to capture information about the stored procedure activity and all of the activities nested within it.

An event monitor for DB2 activities named DB2ACTIVITIES has been activated. The administrator uses the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES function to obtain the application handle, unit of work ID and activity ID for the call of this stored procedure. Assuming that the activity is identified by an

application handle of 1, a unit of work ID of 2 and an activity ID of 3, the administrator can now issue the call to WLM_CAPTURE_ACTIVITY_IN_PROGRESS as follows:

```
CALL WLM_CAPTURE_ACTIVITY_IN_PROGRESS(1,2,3)
```

After the procedure is completed, the administrator can use the following table function to find out where the activity spent its time. The function retrieves the information from the DB2ACTIVITIES event monitor.

```
CREATE FUNCTION SHOWCAPTUREDACTIVITY(APPHNDL BIGINT,
                                     UOWID INTEGER,
                                     ACTIVITYID INTEGER)
  RETURNS TABLE (UOW_ID INTEGER, ACTIVITY_ID INTEGER, STMT_TEXT VARCHAR(40),
    LIFE_TIME DOUBLE)
  LANGUAGE SQL
  READS SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN WITH RAH (LEVEL, APPL_ID, PARENT_UOW_ID, PARENT_ACTIVITY_ID,
          UOW_ID, ACTIVITY_ID, STMT_TEXT, ACT_EXEC_TIME) AS
  (SELECT 1, ROOT.APPL_ID, ROOT.PARENT_UOW_ID,
          ROOT.PARENT_ACTIVITY_ID, ROOT.UOW_ID, ROOT.ACTIVITY_ID,
          ROOTSTMT.STMT_TEXT, ACT_EXEC_TIME
   FROM ACTIVITY_DB2ACTIVITIES ROOT, ACTIVITYSTMT_DB2ACTIVITIES ROOTSTMT
   WHERE ROOT.APPL_ID = ROOTSTMT.APPL_ID AND ROOT.AGENT_ID = APPHNDL
     AND ROOT.UOW_ID = ROOTSTMT.UOW_ID AND ROOT.UOW_ID = UOWID
     AND ROOT.ACTIVITY_ID = ROOTSTMT.ACTIVITY_ID AND ROOT.ACTIVITY_ID = ACTIVITYID
  UNION ALL
   SELECT PARENT.LEVEL +1, CHILD.APPL_ID, CHILD.PARENT_UOW_ID,
          CHILD.PARENT_ACTIVITY_ID, CHILD.UOW_ID,
          CHILD.ACTIVITY_ID, CHILDSTMT.STMT_TEXT, CHILD.ACT_EXEC_TIME
   FROM RAH PARENT, ACTIVITY_DB2ACTIVITIES CHILD,
        ACTIVITYSTMT_DB2ACTIVITIES CHILDSTMT
   WHERE PARENT.APPL_ID = CHILD.APPL_ID AND
         CHILD.APPL_ID = CHILDSTMT.APPL_ID AND
         PARENT.UOW_ID = CHILD.PARENT_UOW_ID AND
         CHILD.UOW_ID = CHILDSTMT.UOW_ID AND
         PARENT.ACTIVITY_ID = CHILD.PARENT_ACTIVITY_ID AND
         CHILD.ACTIVITY_ID = CHILDSTMT.ACTIVITY_ID AND
         PARENT.LEVEL < 64
  )
SELECT UOW_ID, ACTIVITY_ID, SUBSTR(STMT_TEXT,1,40),
               ACT_EXEC_TIME AS
               LIFE_TIME
FROM RAH
```

The following sample query uses the table function:

```
SELECT * FROM TABLE(SHOWCAPTUREDACTIVITY(1, 2, 3))
  AS ACTS ORDER BY UOW_ID, ACTIVITY_ID
```

Usage notes

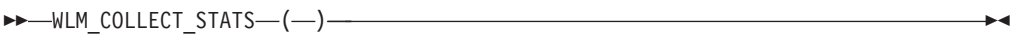
If there is no active activities event monitor, an SQL1633W with SQLSTATE 01H53 is returned.

Activity information is collected only on the coordinator partition for the activity.

WLM_COLLECT_STATS - Collect and reset workload management statistics

The WLM_COLLECT_STATS procedure gathers statistics for service classes, workloads, work classes, and threshold queues and writes them to the statistics event monitor. The procedure also resets the statistics for service classes, workloads, work classes, and threshold queues. If there is no active statistics event monitor, the procedure only resets the statistics.

Syntax



The schema is SYSPROC.

Authorization

EXECUTE privilege on the WLM_COLLECT_STATS procedure.

Examples

Example 1: Call WLM_COLLECT_STATS to collect and reset statistics.

```
CALL WLM_COLLECT_STATS()
```

The following is an example of output from this query.

```
Return Status = 0
```

Example 2: Call WLM_COLLECT_STATS to collect and reset statistics while another call is in progress.

```
CALL WLM_COLLECT_STATS()
```

The following is an example of output from this query.

```
SQL1632W The collect and reset statistics request was ignored because
another collect and reset statistics request is already in progress.
```

Usage notes

The WLM_COLLECT_STATS procedure performs the same collection operation (send statistics to the active statistics event monitor) and reset operation that occur automatically on the interval defined by the **wlm_collect_int** database configuration parameter.

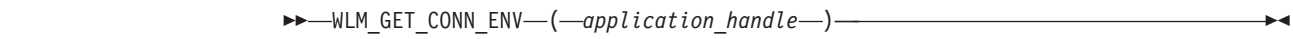
If you call the procedure while another collection and reset request is in progress (for example, while another invocation of the procedure is running or automatic collection is occurring), SQL1632W with SQLSTATE 01H53 is returned, and your new request is ignored.

The WLM_COLLECT_STATS procedure only starts the collection and reset process. The procedure might return to the caller before all statistics have been written to the active statistics event monitor. Depending on how quickly the statistics collection and reset occur, the call to the WLM_COLLECT_STATS procedure (which is itself an activity) is counted in the statistics for either the prior collection interval or the new collection interval that has just started.

WLM_GET_CONN_ENV - get settings for activity data collection for a connection

The WLM_GET_CONN_ENV table function returns for a particular connection the values of settings that control collection of activity data and section actuals. You can use this table function to check the current values of the settings applied by the WLM_SET_CONN_ENV stored procedure.

Syntax



Parameters

application_handle
An input argument of type BIGINT that specifies the application handle for the connection for which information is to be returned. You can use a value of NULL to indicate the connection on which the procedure was invoked.

Authorization

EXECUTE privilege on the WLM_GET_CONN_ENV table function.

Example

The following query checks whether activities are being collected for the current connection.

```
SELECT application_handle,
       xmlparse(document details preserve whitespace)
FROM TABLE (
   WLM_GET_CONN_ENV(
      cast(NULL as bigint))
   ) connenv
```

The following is an example of output from this query.

```
APPLICATION_HANDLE   DETAILS
-------------------- ------------------------------------------------------
                   7 <wlm_conn_env
                     xmlns=http://www.ibm.com/xmlns/prod/db2/mon
                     release="9070100">
                     <collectactdata>NONE</collectactdata>
                     <collectactpartition>COORDINATOR</collectactpartition>
                     <collectsectionactuals>NONE</collectsectionactuals>
                     </wlm_conn_env>
```

Usage notes

The WLM_GET_CONN_ENV table function provides maximum flexibility for formatting output because it returns WLM environment information for a connection as an XML document. The output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function.

The schema for the XML document that is returned in the DETAILS column is available in the file `sqllib/misc/DB2MonRoutines.xsd`. Further details can be found in the file `sqllib/misc/DB2MonCommon.xsd`.

Information returned

Table 238. Information returned for WLM_GET_CONN_ENV

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
DETAILS	BLOB(8M)	XML document that contains connection environment details. See Table 239 for a description of the elements in this document.

Detailed settings returned

Table 239. Detailed metrics returned for WLM_GET_CONN_ENV

Element name	Data type	Description or corresponding monitor element
collectactdata	xs:string(255)	Specifies what kind of activity data is being collected, if any. Possible values are: • NONE • WITHOUT DETAILS • WITH DETAILS • WITH DETAILS, SECTION • WITH DETAILS, SECTION AND VALUES • WITH DETAILS AND VALUES See information about the WLM_SET_CONN_ENV procedure for details about these options.
collectactpartition	xs:string(255)	Specifies where activity data is being collected. Possible values are: • COORDINATOR • ALL See information about the WLM_SET_CONN_ENV procedure for details about these options.
collectsectionactuals	xs:string(255)	Specifies whether section actuals are being collected. Possible values include: • NONE • BASE See information about the WLM_SET_CONN_ENV procedure for details about these options.

WLM_GET_QUEUE_STATS table function - Return threshold queue statistics

The WLM_GET_QUEUE_STATS function returns basic statistics for one or more threshold queues on all active partitions. This function returns one row of statistics for each threshold queue.

Syntax

The schema is SYSPROC.

Table function parameters

threshold_predicate
An input argument of type VARCHAR(27) that specifies a threshold predicate. The possible values are as follows:

CONCDBC
Concurrent database coordinator activities threshold

DBCONN
Total database partition connections threshold

SCCONN
Total service class partition connections threshold

If the argument is null or an empty string, data is returned for all thresholds that meet the other criteria.

The *threshold_predicate* values match those of the THRESHOLDPREDICATE column in the SYSCAT.THRESHOLDS view.

threshold_domain
An input argument of type VARCHAR(18) that specifies a threshold domain. The possible values are as follows:

DB Database

SB Service subclass

SP Service superclass

WA Work action set

If the argument is null or an empty string, data is returned for all thresholds that meet the other criteria.

The *threshold_domain* values match those of the DOMAIN column in the SYSCAT.THRESHOLDS view.

threshold_name
An input argument of type VARCHAR(128) that specifies a threshold name. If the argument is null or an empty string, data is returned for all thresholds that meet the other criteria. The *threshold_name* values match those of the THRESHOLDNAME column in the SYSCAT.THRESHOLDS view.

threshold_id
An input argument of type INTEGER that specifies a threshold ID. If the argument is null or -1, data is returned for all thresholds that meet the other criteria. The *threshold_id* values match those of the THRESHOLDID column in the SYSCAT.THRESHOLDS view.

Authorization

EXECUTE privilege on the WLM_GET_QUEUE_STATS function.

Example

The following query displays the basic statistics for all the queues on a system, across all partitions:

```
SELECT substr(THRESHOLD_NAME, 1, 6) THRESHNAME,
     THRESHOLD_PREDICATE,
     THRESHOLD_DOMAIN,
     DBPARTITIONNUM PART,
     QUEUE_SIZE_TOP,
     QUEUE_TIME_TOTAL,
     QUEUE_ASSIGNMENTS_TOTAL QUEUE_ASSIGN
FROM table(WLM_GET_QUEUE_STATS('', '', '', -1)) as QSTATS
```

Sample output is as follows:

```
THRESHNAME THRESHOLD_PREDICATE         THRESHOLD_DOMAIN   ...
---------- --------------------------- ------------------ ...
LIMIT1     CONCDBC                     DB                 ...
LIMIT2     SCCONN                      SP                 ...
LIMIT3     DBCONN                      DB                 ...

... PART QUEUE_SIZE_TOP QUEUE_TIME_TOTAL QUEUE_ASSIGN
... ---- -------------- ---------------- ------------
... 0              12          1238540          734
... 0               4           741249           24
... 0               7           412785          128
```

Usage note

The function does not aggregate data across queues (on a partition) or across partitions (for one or more queues). However, you can use SQL queries to aggregate data, as shown in the previous example.

Information returned

Table 240. Information returned for WLM_GET_QUEUE_STATS

Column name	Data type	Description
THRESHOLD_PREDICATE	VARCHAR(27)	Threshold predicate of the threshold responsible for this queue. The possible values are as follows: *CONCDBC* Concurrent database coordinator activities threshold *DBCONN* Total database partition connections threshold *SCCONN* Total service class partition connections threshold The threshold predicate values match those of the THRESHOLDPREDICATE column in the SYSCAT.THRESHOLDS view.

Table 240. Information returned for WLM_GET_QUEUE_STATS (continued)

Column name	Data type	Description
THRESHOLD_DOMAIN	VARCHAR(18)	Domain of the threshold responsible for this queue. The possible values are as follows: *DB* Database *SB* Service subclass *SP* Service superclass *WA* Work action set The threshold domain values match those of the DOMAIN column in the SYSCAT.THRESHOLDS view.
THRESHOLD_NAME	VARCHAR(128)	Unique name of the threshold responsible for this queue. The threshold name value matches that of the THRESHOLDNAME column in the SYSCAT.THRESHOLDS view.
THRESHOLD_ID	INTEGER	Unique ID of the threshold responsible for this queue. The threshold ID value matches that of the THRESHOLDID column in the SYSCAT.THRESHOLDS view.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass that is the domain for the threshold responsible for this queue. The value of the column is null if the domain of the threshold is not a service superclass or service subclass.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass that is the domain for the threshold responsible for this queue. The value of the column is null if the domain of the threshold is not a service subclass.
WORK_ACTION_SET_NAME	VARCHAR(128)	Name of the work action set that is the domain for the threshold responsible for this queue. The value of the column is null if the domain of the threshold is not a work action set.
WORK_CLASS_NAME	VARCHAR(128)	Name of the work class whose work action belongs to the work action set that is the domain for the threshold responsible for this queue. The value of the column is null if the domain of the threshold is not a work action set.
WORKLOAD_NAME	VARCHAR(128)	Name of the workload that is the domain for the threshold responsible for this queue. The value of the column is null if the domain of the threshold is not a workload.

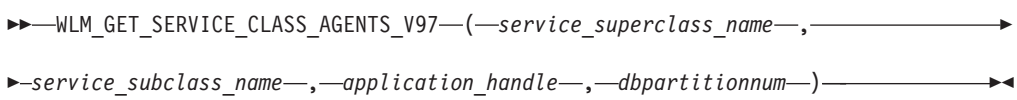
Table 240. Information returned for WLM_GET_QUEUE_STATS (continued)

Column name	Data type	Description
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that trigger a reset of statistics: • You call the WLM_COLLECT_STATS procedure. • The **wlm_collect_int** configuration parameter causes a collection and reset. • You reactivate the database. • You modify the threshold for which queue statistics are being reported and commit the change. The LAST_RESET time stamp is in local time.
QUEUE_SIZE_TOP	INTEGER	Highest number of connections or activities in the queue since the last reset.
QUEUE_TIME_TOTAL	BIGINT	Sum of the times spent in the queue for all connections or activities placed in this queue since the last reset. Units are milliseconds.
QUEUE_ASSIGNMENTS_TOTAL	BIGINT	Number of connections or activities that were assigned to this queue since the last reset.
QUEUE_SIZE_CURRENT	INTEGER	Number of connections or activities in the queue.
QUEUE_TIME_LATEST	BIGINT	Time spent in the queue by the last connection or activity to leave the queue. Units are milliseconds.
QUEUE_EXIT_TIME_LATEST	TIMESTAMP	Time that the last connection or activity left the queue.
THRESHOLD_CURRENT_CONCURRENCY	INTEGER	Number of connections or activities that are currently running according to the threshold.
THRESHOLD_MAX_CONCURRENCY	INTEGER	Maximum number of connections or activities that the threshold allows to be concurrently running.

WLM_GET_SERVICE_CLASS_AGENTS_V97 table function - List agents running in a service class

The WLM_GET_SERVICE_CLASS_AGENTS_V97 function returns the list of agents, fenced mode processes (db2fmp processes), and system entities on a specified partition that are running in a specified service class or on behalf of a specified application. The system entities are non-agent threads and processes, such as page cleaners and prefetchers.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, data is retrieved for all the superclasses in the database.

service_subclass_name
An input argument of type VARCHAR(128) that refers to a specific subclass within a superclass. If the argument is null or an empty string, data is retrieved for all the subclasses in the database.

application_handle
An input argument of type BIGINT that specifies the application handle for which agent information is to be returned. If the argument is null, data is retrieved for all applications in the database. An application handle of 0 returns the system entities only.

dbpartitionnum
An input argument of type INTEGER that specifies the partition number in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If a null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_CLASS_AGENTS_V97 function.

Example

Example 1

The following query returns a list of agents that are associated with application handle 1 for all database partitions. You can determine the application handle by using the LIST APPLICATIONS command or the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 table function.

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
  SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
  SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
  SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
  SUBSTR(AGENT_STATE,1,10) AS AGENTSTATE,
  SUBSTR(REQUEST_TYPE,1,12) AS REQTYPE,
  SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
  SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS_V97(CAST(NULL AS VARCHAR(128)),
  CAST(NULL AS VARCHAR(128)), 1, -2)) AS SCDETAILS
ORDER BY APPHANDLE, PART, AGENT_TID
```

Sample output is as follows:

```
APPHANDLE PART AGENT_TID AGENTTYPE   AGENTSTATE REQTYPE        UOW_ID ACT_ID
--------- ---- --------- ----------- ---------- -------------- ------ ------
1         0    3         COORDINATOR ACTIVE     FETCH          1      5
1         0    4         SUBAGENT    ACTIVE     SUBSECTION:1   1      5
1         1    2         SUBAGENT    ACTIVE     SUBSECTION:2   1      5
```

The output shows a coordinator agent and a subagent on partition 0 and a subagent on partition 1 operating on behalf of an activity with UOW ID 1 and activity ID 5. The AGENTTYPE column with a value of COORDINATOR has a value of FETCH for the REQTYPE column (which indicates the main or initial request type). This means that the type of request is a fetch request for the coordinator agent.

Example 2

The following query determines which lock an agent is waiting on:

```
db2 select event_object, event_type, event_state, varchar(event_object_name, 30)
  as event_object_name
  from table(wlm_get_service_class_agents_v97('','',cast(NULL as bigint), -1)) as t
```

Sample output is as follows:

```
EVENT_OBJECT    EVENT_TYPE        EVENT_STATE         EVENT_OBJECT_NAME
--------------- ----------------- ------------------- --------------------------
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         WAIT              IDLE                -
LOCK            ACQUIRE           IDLE                02000500000000000000000054
ROUTINE         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -

  21 record(s) selected.
```

Using the same query at a later time shows that the WLM threshold has queued an agent:

```
EVENT_OBJECT    EVENT_TYPE        EVENT_STATE         EVENT_OBJECT_NAME
--------------- ----------------- ------------------- --------------------------
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
WLM_QUEUE       WAIT              IDLE                MYCONCDBCOORDTH
ROUTINE         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
REQUEST         PROCESS           EXECUTING           -
```

```
REQUEST         PROCESS           EXECUTING          -
REQUEST         PROCESS           EXECUTING          -
REQUEST         PROCESS           EXECUTING          -
REQUEST         PROCESS           EXECUTING          -
REQUEST         PROCESS           EXECUTING          -
REQUEST         PROCESS           EXECUTING          -
REQUEST         PROCESS           EXECUTING          -
REQUEST         PROCESS           EXECUTING          -

  21 record(s) selected.
```

Usage note

The parameters are, in effect, ANDed together. That is, if you specify conflicting input parameters, such as a service superclass SUP_A and a subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows are returned.

Information returned

Table 241. Information returned by WLM_GET_SERVICE_CLASS_AGENTS_V97

Column name	Data type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR (128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR (128)	Name of the service subclass from which this record was collected.
APPLICATION_HANDLE	BIGINT	System-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16-bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16-bit counter. In addition, this identifier is the same on every partition where the application makes a secondary connection.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
ENTITY	VARCHAR (32)	One of the following values: • If the type of entity is an agent, the value is db2agent. • If the type of entity is a fenced mode process, the value is db2fmp (*pid*) where *pid* is the process ID of the fenced mode process. • Otherwise, the value is the name of the system entity.
WORKLOAD_NAME	VARCHAR (128)	Name of the workload from which this record was collected.
WORKLOAD_OCCURRENCE_ID	INTEGER	ID of the workload occurrence. This ID does not uniquely identify the workload occurrence unless it is coupled with the coordinator database partition number and the workload name.
UOW_ID	INTEGER	Unique ID of the unit of work that this activity started in.
ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work.
PARENT_UOW_ID	INTEGER	Unique ID of the unit of work that the parent activity of the activity started in. The value of the column is null if this activity has no parent.
PARENT_ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work for the parent of the activity whose ID is the same as activity_id. The value of this column is null if this activity has no parent.

Table 241. Information returned by WLM_GET_SERVICE_CLASS_AGENTS_V97 (continued)

Column name	Data type	Description
AGENT_TID	BIGINT	Thread ID of the agent or system entity. If this ID is unavailable, the value of the column is null.
AGENT_TYPE	VARCHAR (32)	Agent type. The agent types are as follows: • COORDINATOR • OTHER • PDBSUBAGENT • SMPSUBAGENT If the value is COORDINATOR, the agent ID might change in concentrator environments.
SMP_COORDINATOR	INTEGER	Indication of whether the agent is an SMP coordinator: 1 for yes and 0 for no.
AGENT_SUBTYPE	VARCHAR (32)	Agent subtype. The possible subtypes are as follows: • DSS • OTHER • RPC • SMP
AGENT_STATE	VARCHAR (32)	Indication of whether an agent is associated or active. The possible values are: • ASSOCIATED • ACTIVE
EVENT_TYPE	VARCHAR (32)	Type of event last processed by this agent. The possible values are as follows: • ACQUIRE • PROCESS • WAIT See Table 242 on page 923 for more information about possible values for this column.
EVENT_OBJECT	VARCHAR (32)	Object of the event last processed by this agent. The possible values are as follows: • COMPRESSION_DICTIONARY_BUILD • IMPLICIT_REBIND • INDEX_RECREATE • LOCK • LOCK_ESCALATION • QP_QUEUE • REMOTE_REQUEST • REQUEST • ROUTINE • WLM_QUEUE See Table 242 on page 923 for more information about possible values for this column.

Table 241. Information returned by WLM_GET_SERVICE_CLASS_AGENTS_V97 (continued)

Column name	Data type	Description
EVENT_STATE	VARCHAR (32)	State of the event last processed by this agent. The possible values are as follows: • EXECUTING • IDLE See Table 242 on page 923 for more information about possible values for this column.
REQUEST_ID	VARCHAR (64)	Request ID. This value is unique only in combination with the value of *application_handle*. You can use this combination to distinguish between one request that is taking a long time and multiple requests; for example, to distinguish between one long fetch and multiple fetches.
REQUEST_TYPE	VARCHAR (32)	Type of request. The possible values are as follows: • For coordinator agents: – CLOSE – COMMIT – COMPILE – DESCRIBE – EXCSQLSET – EXECIMMD – EXECUTE – FETCH – INTERNAL *number*, where *number* is the value of the internal constant – OPEN – PREPARE – REBIND – REDISTRIBUTE – REORG – ROLLBACK – RUNSTATS • For subagents with an AGENT_SUBTYPE of DSS or SMP: – If the subsection number is nonzero, the subsection number in the form SUBSECTION:*subsection number*; otherwise, returns NULL.

Table 241. Information returned by WLM_GET_SERVICE_CLASS_AGENTS_V97 (continued)

Column name	Data type	Description
REQUEST_TYPE (continued)	VARCHAR (32)	• For subagents with an AGENT_SUBTYPE of RPC: – ABP – CATALOG – INTERNAL – REORG – RUNSTATS – WLM • For subagents with a SUBTYPE of OTHER: – ABP – APP_RBSVPT – APP_RELSVPT – BACKUP – CLOSE – EXTERNAL_RBSVPT – EVMON – FORCE – FORCE_ALL – INTERNAL *number*, where *number* is the value of the internal constant – INTERRUPT – NOOP (if there is no request) – QP – REDISTRIBUTE – STMT_RBSVPT – STOP_USING – UPDATE_DBM_CFG – WLM
NESTING_LEVEL	INTEGER	Nesting level of the activity whose ID is activity_id. Nesting level is the depth to which this activity is nested within its topmost parent activity.
INVOCATION_ID	INTEGER	An identifier that distinguishes one invocation of a routine from others at the same nesting level within a unit of work. It is unique within a unit of work for a specific nesting level.
ROUTINE_ID	INTEGER	Unique ID for a routine. The value of this column is null if the activity is not part of a routine.
EVENT_OBJECT_NAME	VARCHAR (1024)	Event object name. If the value of EVENT_OBJECT is LOCK, the value of this column is the name of the lock that the agent is waiting on. If the value of EVENT_OBJECT is WLM_QUEUE, the value of the column is the name of the WLM threshold that the agent is queued on. Otherwise, the value is NULL. See Table 242 on page 923 for more information about possible values for this column.
APPLICATION_NAME	VARCHAR (128)	appl_name - Application name
APPLICATION_ID	VARCHAR (128)	appl_id - Application ID

Table 241. Information returned by WLM_GET_SERVICE_CLASS_AGENTS_V97 (continued)

Column name	Data type	Description
CLIENT_PID	BIGINT	client_pid - Client process ID
SESSION_AUTH_ID	VARCHAR (128)	session_auth_id - Session authorization ID
REQUEST_START_TIME	TIMESTAMP	Time that the agent started processing the request on which it is currently working
AGENT_STATE_LAST _UPDATE_TIME	TIMESTAMP	The last time that the event, being processed by the agent, was changed. The event currently processed by the agent is identified by the EVENT_TYPE, EVENT_OBJECT, and EVENT_STATE columns.
EXECUTABLE_ID	VARCHAR (32) FOR BIT DATA	Binary token generated on the data server that uniquely identifies the section that an agent is working on. You can use the executable ID as input to different monitoring interfaces to obtain data about the section. A NULL value is returned if the agent is not working on a section.

Note: The possible combinations of EVENT_STATE, EVENT_TYPE, EVENT_OBJECT and EVENT_OBJECT_NAME column values are listed in the following table.

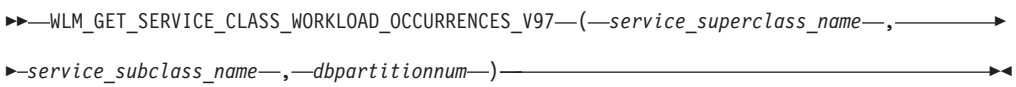
Table 242. Possible combinations for EVENT_STATE, EVENT_TYPE, EVENT_OBJECT and EVENT_OBJECT_NAME column values

Event description	EVENT_STATE value	EVENT_TYPE value	EVENT_OBJECT value	EVENT_OBJECT_NAME value
Acquire lock	IDLE	ACQUIRE	LOCK	Lock name
Escalate lock	EXECUTING	PROCESS	LOCK_ESCALATION	NULL
Process request	EXECUTING	PROCESS	REQUEST	NULL
Wait for a new request	IDLE	WAIT	REQUEST	NULL
Wait for a request to be processed at a remote partition	IDLE	WAIT	REMOTE_REQUEST	NULL
Wait on a Query Patroller queue	IDLE	WAIT	QP_QUEUE	NULL
Wait on a WLM threshold queue	IDLE	WAIT	WLM_QUEUE	Threshold name
Process a routine	EXECUTING	PROCESS	ROUTINE	NULL
Recreate an index	EXECUTING	PROCESS	INDEX_RECREATE	NULL
Build compression dictionary	EXECUTING	PROCESS	COMP_DICT_BLD	NULL
Implicit rebind	EXECUTING	PROCESS	IMPLICIT_REBIND	NULL

WLM_GET_SERVICE_CLASS_WORKLOAD _OCCURRENCES_V97 - List workload occurrences

The WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 function returns the list of all workload occurrences running in a specified service class on a particular partition. A workload occurrence is a specific database connection whose attributes match the definition of a workload and hence is associated with or assigned to the workload.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, the data is retrieved for all the superclasses in the database that match the values of the other parameters.

service_subclass_name
Target service subclass for the workload occurrence. Any work submitted by this workload occurrence will run in this service subclass under the target service superclass with the exception of activities that are mapped, or remapped, to a different subclass.

dbpartitionnum
An input argument of type INTEGER that specifies the number of a partition in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 function.

Example

If an administrator wants to see what workload occurrences are running on the system as a whole, the administrator can call the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 function by specifying a null value or an empty string for *service_superclass_name* and *service_subclass_name* and -2 for *dbpartitionnum*:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       SUBSTR(CHAR(COORD_PARTITION_NUM),1,4) AS COORDPART,
       SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHNDL,
       SUBSTR(WORKLOAD_NAME,1,22) AS WORKLOAD_NAME,
       SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97
```

```
       (CAST(NULL AS VARCHAR(128)), CAST(NULL AS VARCHAR(128)), -2))
        AS SCINFO
  ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART, APPHNDL,
        WORKLOAD_NAME, WLO_ID
```

If the system has four database partitions and is currently running two workloads, the previous query produces results such as the following ones:

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART COORDPART ...
------------------- ------------------ ---- --------- ...
SYSDEFAULTMAINTENAN SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTSYSTEMCLA SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    0         ...

... APPHNDL WORKLOAD_NAME          WLO_ID
... ------- ---------------------- ------
... -       -                      -
... -       -                      -
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
```

Usage note

The parameters are, in effect, ANDed together. That is, if you specify conflicting input parameters, such as a service superclass SUP_A and a subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows are returned.

Note: Statistics reported for the workload occurrence (for example, coord_act_completed_total) are reset at the beginning of each unit of work when they are combined with the corresponding workload statistics.

Information returned

Table 243. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97

Column name	Data type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
COORD_PARTITION_NUM	SMALLINT	Partition number of the coordinator partition of the specified workload occurrence.

Table 243. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 (continued)

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	System-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16-bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16-bit counter. In addition, this identifier is the same on every partition where the application makes a secondary connection.
WORKLOAD_NAME	VARCHAR(128)	Name of the workload from which this record was collected.
WORKLOAD_OCCURRENCE_ID	INTEGER	ID of the workload occurrence. This ID does not uniquely identify the workload occurrence unless it is coupled with the coordinator database partition number and the workload name.
WORKLOAD_OCCURRENCE_STATE	VARCHAR(32)	Workload occurrence state. The values are as follows: *DECOUPLED* Workload occurrence does not have a coordinator agent assigned (concentrator case). *DISCONNECTPEND* Workload occurrence is disconnecting from the database. *FORCED* Workload occurrence has been forced off the database. *INTERRUPTED* Workload occurrence has been interrupted. *QUEUED* Workload occurrence coordinator agent is queued by Query Patroller or a workload management queuing threshold. In a partitioned database environment, this state might indicate that the coordinator agent has made an RPC to the catalog partition to obtain threshold tickets and has not yet received a response. *TRANSIENT* Workload occurrence has not yet been mapped to a service superclass. *UOWEXEC* Workload occurrence is processing a request. *UOWWAIT* Workload occurrence is waiting for a request from the client.

Table 243. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 (continued)

Column name	Data type	Description
UOW_ID	INTEGER	Unique ID of the unit of work that this workload occurrence started in.
SYSTEM_AUTH_ID	VARCHAR(128)	System authorization ID under which the workload occurrence was inserted into the system.
SESSION_AUTH_ID	VARCHAR(128)	Session authorization ID under which the workload occurrence was inserted into the system.
APPLICATION_NAME	VARCHAR(128)	Name of the application that created this workload occurrence.
CLIENT_WRKSTNNAME	VARCHAR(255)	Current value of the CLIENT_WRKSTNNAME special register for this workload occurrence.
CLIENT_ACCTNG	VARCHAR(255)	Current value of the CLIENT_ACCTNG special register for this workload occurrence.
CLIENT_USER	VARCHAR(255)	Current value of the CLIENT_USERID special register for this workload occurrence.
CLIENT_APPLNAME	VARCHAR(255)	Current value of the CLIENT_APPLNAME special register for this workload occurrence.
COORD_ACT_COMPLETED_TOTAL	INTEGER	Number of coordinator activities at any nesting level that were completed so far in the current unit of work of this workload occurrence. This statistic is updated every time that an activity in this workload occurrence is completed and is reset at the beginning of each unit of work.
COORD_ACT_ABORTED_TOTAL	INTEGER	Number of coordinator activities that were aborted so far in the current unit of work of this workload occurrence. This statistic is updated every time that an activity in this workload occurrence is aborted and is reset at the beginning of each unit of work.
COORD_ACT_REJECTED_TOTAL	INTEGER	Number of coordinator activities that were rejected so far in the current unit of work of this workload occurrence. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action or a predictive threshold. This statistic is updated every time that an activity in this workload occurrence is rejected and is reset at the beginning of each unit of work.
CONCURRENT_ACT_TOP	INTEGER	Highest number of concurrent activities at any nesting level in either executing state (which includes idle and waiting) or queued state that has been reached for this workload occurrence in the current unit of work. This statistic is reset at the beginning of each unit of work.
ADDRESS	VARCHAR(255)	IP address or secure domain name that created this workload occurrence. Secure domain names are shown converted to IP addresses.

WLM_GET_SERVICE_SUBCLASS_STATS_V97 table function - Return statistics of service subclasses

The WLM_GET_SERVICE_SUBCLASS_STATS_V97 function returns basic statistics for one or more service subclasses.

Syntax

The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, the data is retrieved for all of the superclasses in the database.

service_subclass_name
An input argument of type VARCHAR(128) that specifies the name of a service subclass in the currently connected database. If the argument is null or an empty string, the data is retrieved for all of the subclasses in the database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_SUBCLASS_STATS_V97 function.

Examples

Example 1: Because every activity must be mapped to a DB2 service class before being run, you can monitor the global state of the system by using the service class statistics table functions and querying all of the service classes on all partitions. In the following example, a null value is passed for *service_superclass_name* and *service_subclass_name* to return statistics for all service classes, and the value -2 is specified for *dbpartitionnum* to return statistics for all partitions:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
      SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
      SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
      CAST(COORD_ACT_LIFETIME_AVG / 1000 AS DECIMAL(9,3))
        AS AVGLIFETIME,
      CAST(COORD_ACT_LIFETIME_STDDEV / 1000 AS DECIMAL(9,3))
        AS STDDEVLIFETIME,
      SUBSTR(CAST(LAST_RESET AS VARCHAR(30)),1,16) AS LAST_RESET
 FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS_V97(CAST(NULL AS VARCHAR(128)),
      CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
 ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART
```

The statement returns service class statistics such as average activity lifetime and standard deviation in seconds, as shown in the following sample output:

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART ...
------------------- ------------------ ---- ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    ...

... AVGLIFETIME STDDEVLIFETIME LAST_RESET
... ----------- -------------- ----------------
...     691.242         34.322 2006-07-24-11.44
...     644.740         22.124 2006-07-24-11.44
...     612.431         43.347 2006-07-24-11.44
...     593.451         28.329 2006-07-24-11.44
```

Example 2: The same table function can also give the highest value for average concurrency of coordinator activities running in the service class on each partition:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       CONCURRENT_ACT_TOP AS ACTTOP,
       CONCURRENT_WLO_TOP AS CONNTOP
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS_V97(CAST(NULL AS VARCHAR(128)),
       CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART
```

Sample output is as follows:

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART ACTTOP    CONNTOP
------------------- ------------------ ---- --------- ---------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0           10         7
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1            0         0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2            0         0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3            0         0
```

By checking the average execution times and numbers of activities in the output of this table function, you can get a good high-level view of the load on each partition for a specific database. Any significant variations in the high-level gauges returned by this table function might indicate a change in the load on the system.

Example 3: If an activity uses thresholds with REMAP ACTIVITY TO actions, the activity might spend time in more than one service class during its lifetime. You can determine how many activities have passed through a set of service classes by looking at the ACTIVITIES_MAPPED_IN and ACTIVITIES_MAPPED_OUT columns, as shown in the following example:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       ACTIVITIES_MAPPED_IN AS MAPPED_IN,
       ACTIVITIES_MAPPED_OUT AS MAPPED_OUT
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS_V97(CAST(NULL AS VARCHAR(128)),
       CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME
```

Sample output is as follows:

```
SUPERCLASS_NAME     SUBCLASS_NAME      MAPPED_IN MAPPED_OUT
------------------- ------------------ --------- ----------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS         0          0
SUPERCLASS1         SYSDEFAULTSUBCLASS         0          0
SUPERCLASS1         SUBCLASS1                  0          7
SUPERCLASS1         SUBCLASS2                  7          0
```

Usage notes

Some statistics are returned only if you set the COLLECT AGGREGATE ACTIVITY DATA and COLLECT AGGREGATE REQUEST DATA parameters for the corresponding service subclass to a value other than NONE.

The WLM_GET_SERVICE_SUBCLASS_STATS_V97 table function returns one row of data per service subclass and per partition. The function does not aggregate data across service classes (on a partition) or across partitions (for one or more service classes). However, you can use SQL queries to aggregate data.

The parameters are, in effect, ANDed together. That is, if you specify conflicting input parameters, such as a superclass named SUPA and a subclass named SUBB such that SUBB is not a subclass of SUPA, no rows are returned.

Information returned

Table 244. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS_V97

Column name	Data type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that trigger a reset of statistics: • You call the WLM_COLLECT_STATS procedure. • The **wlm_collect_int** configuration parameter causes a collection and reset. • You reactivate the database. • You modify the service subclass for which statistics are being reported and commit the change. The LAST_RESET time stamp is in local time.
COORD_ACT_COMPLETED_TOTAL	BIGINT	The total number of coordinator activities that were submitted since the last reset and that were completed successfully. This count is updated as each activity is completed. If you remap an activity to a different service subclass, that activity counts only toward the total of the subclass in which it is completed.

Table 244. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS_V97 (continued)

Column name	Data type	Description
COORD_ACT_ABORTED_TOTAL	BIGINT	The total number of coordinator activities that were submitted since the last reset and that were completed with errors. This count is updated as each activity aborts. If you remap an activity to a different service subclass, that activity counts only toward the total of the subclass in which it aborts.
COORD_ACT_REJECTED_TOTAL	BIGINT	The total number of coordinator activities that were submitted since the last reset and that were rejected before execution. Activities are counted as rejected when they are prevented from running by either a prevent execution work action or a predictive threshold. This count is updated as each activity is rejected.
CONCURRENT_ACT_TOP	INTEGER	Highest number of concurrent activities at any nesting level in executing state (which includes idle and waiting) that has been reached for this service subclass.
COORD_ACT_LIFETIME_TOP	BIGINT	High watermark for coordinator activity lifetime, evaluated over all nesting levels. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds. To use this statistic effectively when the service class includes remapped subclasses, you must aggregate the COORD_ACT_LIFETIME_TOP high watermark of the service subclass with that of other subclasses affected by the same remapping threshold or thresholds. You must aggregate these values because an activity can be completed after the subclass has been remapped to a different service subclass. The time that the activity spends in other service subclasses before being remapped is counted only toward the service class in which it is completed.

Table 244. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS_V97 (continued)

Column name	Data type	Description
COORD_ACT_LIFETIME_AVG	DOUBLE	Arithmetic mean of lifetime for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds. The COORD_ACT_LIFETIME_AVG value of a service subclass is unaffected by activities that pass through the subclass but are remapped to a different subclass before they are completed.
COORD_ACT_LIFETIME_STDDEV	DOUBLE	Standard deviation of lifetime for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds. This standard deviation is computed from the coordinator activity lifetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The COORD_ACT_LIFETIME_STDDEV value of a service subclass is unaffected by activities that pass through the service subclass but are remapped to a different subclass before they are completed.
COORD_ACT_EXEC_TIME_AVG	DOUBLE	Arithmetic mean of the execution times for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds. The execution time average of a service subclass is unaffected by activities that pass through the subclass but are remapped to a different subclass before they are completed.

Table 244. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS_V97 (continued)

Column name	Data type	Description
COORD_ACT_EXEC_TIME_STDDEV	DOUBLE	Standard deviation of the execution times for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. Units are milliseconds. This standard deviation is computed from the coordinator activity executetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The execution time standard deviation of a service subclass is unaffected by activities that pass through the subclass but are remapped to a different subclass before they are completed.
COORD_ACT_QUEUE_TIME_AVG	DOUBLE	Arithmetic mean of the queue time for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds. The queue time average is counted only toward the service subclass in which the activity was queued.
COORD_ACT_QUEUE_TIME_STDDEV	DOUBLE	Standard deviation of the queue time for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds. This standard deviation is computed from the coordinator activity queuetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The queue time standard deviation is counted only toward the service subclass in which the activity was queued.
NUM_REQUESTS_ACTIVE	BIGINT	Number of requests that are running in the service subclass at the time that this table function is running.

Table 244. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS_V97 (continued)

Column name	Data type	Description
NUM_REQUESTS_TOTAL	BIGINT	Number of requests that finished running in this service subclass since the last reset. This finished state applies to any request regardless of its membership in an activity. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. The NUM_REQUESTS_TOTAL value of a service subclass is unaffected by requests that pass through the service subclass but are not completed in it.
REQUEST_EXEC_TIME_AVG	DOUBLE	Arithmetic mean of the execution times for requests that were associated with this service subclass since the last reset. Units are milliseconds. If the internally tracked average has overflowed, the value -2 is returned. If the COLLECT AGGREGATE REQUEST DATA parameter of this service class is set to NONE, the value of this column is NULL. The execution time average of a service subclass is unaffected by requests that pass through the subclass but are not completed in it.
REQUEST_EXEC_TIME_STDDEV	DOUBLE	Standard deviation of the execution times for requests that were associated with this service subclass since the last reset. Units are milliseconds. If the COLLECT AGGREGATE REQUEST DATA parameter of the service class is set to NONE, the value of this column is NULL. This standard deviation is computed from the request executetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The execution time standard deviation of a service subclass is unaffected by requests that pass through the subclass but were not completed in it.

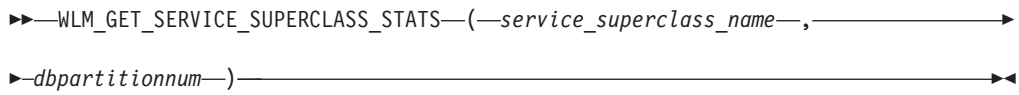
Table 244. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS_V97 (continued)

Column name	Data type	Description
REQUEST_EXEC_TIME_TOTAL	BIGINT	Sum of the execution times for requests that were associated with this service subclass since the last reset. Units are milliseconds. If the COLLECT AGGREGATE REQUEST DATA parameter of the service class is set to NONE, the value of this column is NULL. This total is computed from the request execution time histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The execution time total of a service subclass is unaffected by requests that pass through the subclass but are not completed in it.
ACT_REMAPPED_IN	BIGINT	Number of activities remapped into this service subclass by a threshold REMAP ACTIVITY action since the last reset.
ACT_REMAPPED_OUT	BIGINT	Number of activities remapped out of this service subclass by a threshold REMAP ACTIVITY action since the last reset.
CONCURRENT_WLO_TOP	INTEGER	Highest number of concurrent occurrences of the specified workload on this partition since the last reset.
UOW_TOTAL_TIME_TOP	BIGINT	High watermark for unit of work lifetime, in milliseconds. Returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class is set to NONE. **Note:** Measurements for this high watermark are computed for the service class assigned by the workload. Any mapping by a work action set to change the service class of an activity does not affect this high watermark.

WLM_GET_SERVICE_SUPERCLASS_STATS - Return statistics of service superclasses

The WLM_GET_SERVICE_SUPERCLASS_STATS function returns basic statistics for one or more service superclasses.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, data is retrieved for all the superclasses in the database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_SUPERCLASS_STATS function.

Example

The following query displays the basic statistics for all the service superclasses on the system, across all database partitions:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME, 1, 26) SERVICE_SUPERCLASS_NAME,
       DBPARTITIONNUM,
       LAST_RESET,
       CONCURRENT_CONNECTION_TOP CONCURRENT_CONN_TOP
FROM TABLE(WLM_GET_SERVICE_SUPERCLASS_STATS('', -2)) as SCSTATS
```

Sample output is as follows:

```
SERVICE_SUPERCLASS_NAME    DBPARTITIONNUM ...
-------------------------- -------------- ...
SYSDEFAULTSYSTEMCLASS                   0 ...
SYSDEFAULTMAINTENANCECLASS              0 ...
SYSDEFAULTUSERCLASS                     0 ...

... LAST_RESET                 CONCURRENT_CONN_TOP
... -------------------------- -------------------
... 2006-09-05-09.38.44.396788                   0
... 2006-09-05-09.38.44.396795                   0
... 2006-09-05-09.38.44.396796                   1
```

Usage note

The WLM_GET_SERVICE_SUPERCLASS_STATS table function returns one row of data per service superclass and per partition. The function does not aggregate data across service superclasses (on a partition) or across partitions (for one or more service superclasses). However, you can use SQL queries to aggregate data, as shown in the previous example.

Information returned

Table 245. Information returned for WLM_GET_SERVICE_SUPERCLASS_STATS

Column name	Data type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.

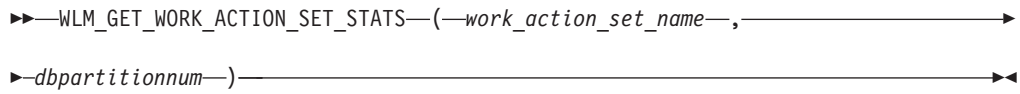
Table 245. Information returned for WLM_GET_SERVICE_SUPERCLASS_STATS (continued)

Column name	Data type	Description
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that trigger a reset of statistics: • You call the WLM_COLLECT_STATS procedure. • The **wlm_collect_int** configuration parameter causes a collection and reset. • You reactivate the database. • You modify the service superclass for which statistics are being reported and commit the change. The LAST_RESET time stamp is in local time.
CONCURRENT_CONNECTION_TOP	INTEGER	Highest number of concurrent coordinator connections in this class since the last reset.

WLM_GET_WORK_ACTION_SET_STATS - Return work action set statistics

The WLM_GET_WORK_ACTION_SET_STATS function returns the statistics for a work action set.

Syntax



The schema is SYSPROC.

Table function parameters

work_action_set_name
An input argument of type VARCHAR(128) that specifies the work action set to return statistics for. If the argument is null or an empty string, statistics are returned for all work action sets.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_WORK_ACTION_SET_STATS function.

Example

Assume that there are three work classes: ReadClass, WriteClass, and LoadClass. There is a work action associated with ReadClass and a work action associated with LoadClass, but there is no work action associated with WriteClass. On partition 0, the number of activities currently running or queued are as follows:

- ReadClass class: eight

- WriteClass class: four
- LoadClass class: two
- Unassigned: three

```
SELECT SUBSTR(WORK_ACTION_SET_NAME,1,18) AS WORK_ACTION_SET_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       SUBSTR(WORK_CLASS_NAME,1,15) AS WORK_CLASS_NAME,
       LAST_RESET,
       SUBSTR(CHAR(ACT_TOTAL),1,14) AS ACT_TOTAL
  FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS
       (CAST(NULL AS VARCHAR(128)), -2)) AS WASSTATS
  ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME, PART
```

Sample output is as follows. Because there is no work action associated with the WriteClass work class, the four activities to which it applies are counted in the artificial class denoted by an asterisk (*) in the output. The three activities that were not assigned to any work class are also included in the artificial class.

```
WORK_ACTION_SET_NAME PART WORK_CLASS_NAME LAST_RESET                ACT_TOTAL
-------------------- ---- -------------- ------------------------- --------------
AdminActionSet       0    ReadClass      2005-11-25-18.52.49.343000 8
AdminActionSet       1    ReadClass      2005-11-25-18.52.50.478000 0
AdminActionSet       0    LoadClass      2005-11-25-18.52.49.343000 2
AdminActionSet       1    LoadClass      2005-11-25-18.52.50.478000 0
AdminActionSet       0    *              2005-11-25-18.52.49.343000 7
AdminActionSet       1    *              2005-11-25-18.52.50.478000 0
```

Information returned

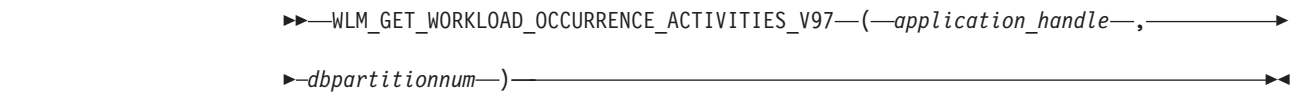
Table 246. Information returned for WLM_GET_WORK_ACTION_SET_STATS

Column name	Data type	Description
WORK_ACTION_SET_NAME	VARCHAR(128)	Name of the work action set. A name is returned only if you enable the work action set.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that trigger a reset of statistics: • You call the WLM_COLLECT_STATS procedure. • The **wlm_collect_int** configuration parameter causes a collection and reset. • You reactivate the database. • You modify the work action set for which statistics are being reported and commit the change. The LAST_RESET time stamp is in local time.
WORK_CLASS_NAME	VARCHAR(128)	Name of the work class related to the specified work action set. A work class name is returned only if you enable a work action associated with the work class. The asterisk (*) represents an artificial work class created to count all those activities that did not belong to the other work classes for which you associated one or more work actions.
ACT_TOTAL	BIGINT	Number of activities at any nesting level that were assigned to the work class specified by WORK_CLASS_NAME.

WLM_GET_WORKLOAD_OCCURRENCE _ACTIVITIES_V97 - Return a list of activities

The WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 function returns the list of all activities that were submitted by the specified application on the specified partition and have not yet been completed.

Syntax



The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies an application handle for which a list of activities is to be returned. If the argument is null, the data is retrieved for all the applications in the database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 function.

Examples

Example 1: Activities currently running with a known application handle

After you identify the application handle, you can look up all the activities currently running in this application. For example, suppose that an administrator wants to list the activities of an application whose application handle, determined by using the LIST APPLICATIONS command, is 1. The administrator runs the following query:

```
SELECT SUBSTR(CHAR(COORD_PARTITION_NUM),1,5) AS COORD,
     SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
     SUBSTR(CHAR(UOW_ID),1,5) AS UOWID,
     SUBSTR(CHAR(ACTIVITY_ID),1,5) AS ACTID,
     SUBSTR(CHAR(PARENT_UOW_ID),1,8) AS PARUOWID,
     SUBSTR(CHAR(PARENT_ACTIVITY_ID),1,8) AS PARACTID,
            ACTIVITY_TYPE AS ACTTYPE,
     SUBSTR(CHAR(NESTING_LEVEL),1,7) AS NESTING
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97(1, -2)) AS WLOACTS
ORDER BY PART, UOWID, ACTID
```

Sample output from the query is as follows:

```
COORD PART UOWID ACTID PARUOWID PARACTID ACTTYPE  NESTING
----- ---- ----- ----- -------- -------- -------- -------
0     0    2     3     -        -        CALL     0
0     0    2     5     2        3        READ_DML 1
```

Example 2: Activities currently running on the system

The following query joins the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 output with the MON_GET_PKG_CACHE_STMT output on EXECUTABLE_ID to provide statement text for all the SQL activities currently running on the system:

```
SELECT t.application_handle,
       t.uow_id,
       t.activity_id,
       varchar(p.stmt_text, 256) as stmt_text
FROM table(wlm_get_workload_occurrence_activities_v97(NULL, -1)) as t,
     table(mon_get_pkg_cache_stmt(NULL, NULL, NULL, -1)) as p
WHERE t.executable_id = p.executable_id
```

Sample output is as follows:

```
APPLICATION_HANDLE   UOW_ID      ACTIVITY_ID    STMT_TEXT
-------------------- ----------- -------------- ------------------------------
1                    1           1              SELECT * FROM SYSCAT.TABLES
47                   1           36             INSERT INTO T1 VALUES(123)
```

Information returned

Table 247. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	System-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16-bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16-bit counter. In addition, this identifier is the same on every partition where the application makes a secondary connection.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
COORD_PARTITION_NUM	SMALLINT	Coordinator partition number of the activity.
LOCAL_START_TIME	TIMESTAMP	Local time that this activity began doing work on the partition. The value of the column is null when an activity has entered the system but is in a queue and has not started running.
UOW_ID	INTEGER	Unique ID of the original unit of work that the activity started in.
ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work.
PARENT_UOW_ID	INTEGER	Unique ID of the original unit of work that the parent activity of the activity started in. The value of this column is null if the activity has no parent activity or is at a remote partition.

Table 247. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 (continued)

Column name	Data type	Description
PARENT_ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work for the parent of the activity whose ID is specified by ACTIVITY_ID. The value of this column is null if the activity has no parent activity or is at a remote partition.
ACTIVITY_STATE	VARCHAR(32)	Activity state. Possible values are as follows: *CANCEL_PENDING* The activity was cancelled because there was no agent actively working on a request for the activity. The next time that a request is submitted as part of the activity, the activity will be cancelled, and an SQL4725N error will be generated. *EXECUTING* Agents are actively working on a request for the activity. *IDLE* There is no agent actively processing a request for the activity. *INITIALIZING* The activity has been submitted but has not yet started running. During the initializing state, predictive thresholds are applied to the activity to determine whether the activity will be allowed to run.

Table 247. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 (continued)

Column name	Data type	Description
ACTIVITY_STATE (continued)	VARCHAR(32)	Activity state. Possible values are as follows: *QP_CANCEL_PENDING* This state is the same as the CANCEL_PENDING state except that the activity was cancelled by Query Patroller rather than by the WLM_CANCEL_ACTIVITY procedure. *QP_QUEUED* The activity is queued by Query Patroller. *QUEUED* The activity is queued by a workload management queuing threshold. In a partitioned database environment, this state might mean that the coordinator agent has made an RPC to the catalog partition to obtain threshold tickets and has not yet received a response. This state might indicate that the activity has been queued by a workload management queuing threshold or, if not much time has elapsed, can indicate that the activity is in the process of obtaining its tickets. To obtain a more accurate picture of whether the activity is being queued, determine what agent is working on the activity, and find out whether the EVENT_OBJECT value of the object at the catalog partition has a value of WLM_QUEUE. *TERMINATING* The activity has finished running and is being removed from the system.
ACTIVITY_TYPE	VARCHAR(32)	Activity type. Possible values are as follows: • CALL • DDL • LOAD • OTHER • READ_DML • WRITE_DML Refer to "Identify types of work with work classes" in *Workload Manager Guide and Reference* for a description of the different types of SQL statements that are associated with each activity type.
NESTING_LEVEL	INTEGER	Depth to which this activity is nested within its topmost parent activity.

Table 247. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 (continued)

Column name	Data type	Description
INVOCATION_ID	INTEGER	An identifier that distinguishes one invocation of a routine from others at the same nesting level within a unit of work. It is unique within a unit of work for a specific nesting level.
ROUTINE_ID	INTEGER	Unique ID of the routine.
UTILITY_ID	INTEGER	One of the following values: • If the activity is a utility, the value is the ID of the utility. • If the activity is not a utility, the value is null.
SERVICE_CLASS_ID	INTEGER	Unique ID of the service class to which this activity belongs.
DATABASE_WORK_ACTION_SET_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of database scope, the value is the ID of the work class set of which this work class is a member. • If this activity has not been categorized into a work class of database scope, the value is null.
DATABASE_WORK_CLASS_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of database scope, the value is the ID of the work class. • If this activity has not been categorized into a work class of database scope, the value is null.
SERVICE_CLASS_WORK_ACTION_SET_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of service class scope, the value is the ID of the work action set associated with the work class set to which the work class belongs. • If this activity has not been categorized into a work class of service class scope, the value is null.
SERVICE_CLASS_WORK_CLASS_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of service class scope, the value is the ID of the work class assigned to this activity. • If this activity has not been categorized into a work class of service class scope, the value is null.

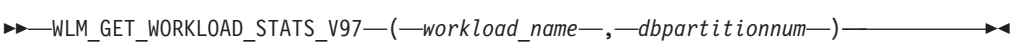
Table 247. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 (continued)

Column name	Data type	Description
EXECUTABLE_ID	VARCHAR(32) FOR BIT DATA	An opaque binary token generated on the data server that uniquely identifies the section. You can use the executable ID as input to different monitoring interfaces to obtain data about the section. For non-SQL activities, such as LOAD, a NULL value is returned.
TOTAL_CPU_TIME	BIGINT	total_cpu_time - Total CPU time
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_RETURNED	BIGINT	rows_returned - Rows returned
QUERY_COST_ESTIMATE	BIGINT	query_cost_estimate - Query cost estimate
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database

WLM_GET_WORKLOAD_STATS_V97 table function - Return workload statistics

The WLM_GET_WORKLOAD_STATS_V97 function returns one row of workload statistics for every combination of workload name and database partition number.

Syntax



The schema is SYSPROC.

Table function parameters

workload_name
An input argument of type VARCHAR(128) that specifies a workload for which the statistics are to be returned. If the argument is NULL or an empty string, statistics are returned for all workloads.

dbpartitionnum
An input argument of type INTEGER that specifies the number of a partition in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If a null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_WORKLOAD_STATS_V97 function.

Example

The following query displays statistics for workloads:

```
SELECT SUBSTR(WORKLOAD_NAME,1,18) AS WL_DEF_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       COORD_ACT_LIFETIME_TOP,
```

```
       COORD_ACT_LIFETIME_AVG,
       COORD_ACT_LIFETIME_STDDEV
FROM TABLE(WLM_GET_WORKLOAD_STATS_V97(CAST(NULL AS VARCHAR(128)), -2)) AS WLSTATS
ORDER BY WL_DEF_NAME, PART
```

Sample output from the query is as follows:

```
WL_DEF_NAME        PART COORD_ACT_LIFETIME_TOP ...
------------------ ---- ---------------------- ...
SYSDEFAULTADMWORKL 0                       -1 ...
SYSDEFAULTUSERWORK 0                       -1 ...
WL1                0                        2 ...

... COORD_ACT_LIFETIME_AVG   COORD_ACT_LIFETIME_STDDEV
... ---------------------- ------------------------
... -1.00000000000000E+000    -1.00000000000000E+000
... -1.00000000000000E+000    -1.00000000000000E+000
... +2.56000000000000E+000    +6.00000000000001E-002
```

Usage note

The function does not aggregate data across workloads, partitions, or service classes. However, you can use SQL queries to aggregate data.

Information returned

Table 248. Information returned by WLM_GET_WORKLOAD_STATS_V97

Column name	Data type	Description
WORKLOAD_NAME	VARCHAR(128)	Name of the workload from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that trigger a reset of statistics: • You call the WLM_COLLECT_STATS procedure. • The **wlm_collect_int** configuration parameter causes a collection and reset. • You reactivate the database. • You modify the workload for which statistics are being reported and commit the change. The LAST_RESET timestamp is in local time.
CONCURRENT_WLO_TOP	INTEGER	Highest number of concurrent occurrences of the specified workload on this partition since the last reset.
CONCURRENT_WLO_ACT_TOP	INTEGER	Highest number of concurrent activities (both coordinator and nested) in either executing state (which includes idle and waiting) or queued state that has been reached in any occurrence of this workload since the last reset. The value of the column is updated by each workload occurrence at the end of its unit of work.
COORD_ACT_COMPLETED_TOTAL	BIGINT	Total number of coordinator activities at any nesting level that were assigned to any occurrence of this workload that were completed since the last reset. The value of this column is updated by each workload occurrence at the end of its unit of work.

Table 248. Information returned by WLM_GET_WORKLOAD_STATS_V97 (continued)

Column name	Data type	Description
COORD_ACT_ABORTED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level that were assigned to any occurrence of this workload that were aborted before completion since the last reset. The value of this column is updated by each workload occurrence at the end of its unit of work.
COORD_ACT_REJECTED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level that were assigned to any occurrence of this workload that were rejected before execution since the last reset. The value of this column is updated by each workload occurrence at the end of its unit of work. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action or a predictive threshold. Unlike the column of the same name in the WLM_GET_SERVICE_SUBCLASS_STATS_V97 function, this WLM_GET_WORKLOAD_STATS_V97 column also includes the number of rejections that occur before an activity can be assigned to a service class. For example, such a rejection occurs when an activity violates the ConcurrentWorkloadOccurrences threshold.
WLO_COMPLETED_TOTAL	BIGINT	Number of workload occurrences to be completed since last reset.
COORD_ACT_LIFETIME_TOP	BIGINT	High watermark for coordinator activity lifetime, collected over all nesting levels. Units are milliseconds. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null.
COORD_ACT_LIFETIME_AVG	DOUBLE	Arithmetic mean of lifetime for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. If the internally tracked average has overflowed, the value -2 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null.
COORD_ACT_LIFETIME_STDDEV	DOUBLE	Standard deviation of lifetime for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null. This standard deviation is computed from the coordinator activity lifetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. If any values fall into the last histogram bin, the value -1 is returned.

Table 248. Information returned by WLM_GET_WORKLOAD_STATS_V97 (continued)

Column name	Data type	Description
COORD_ACT_EXEC_TIME_AVG	DOUBLE	Arithmetic mean of the execution times for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. If the internally tracked average has overflowed, the value -2 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null.
COORD_ACT_EXEC_TIME_STDDEV	DOUBLE	Standard deviation of the execution times for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. This standard deviation is computed from the coordinator activity executetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. If any values fall into the last histogram bin, the value -1 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null.
COORD_ACT_QUEUE_TIME_AVG	DOUBLE	Arithmetic mean of the queue time for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. If the internally tracked average has overflowed, the value -2 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null.
COORD_ACT_QUEUE_TIME_STDDEV	DOUBLE	Standard deviation of the queue time for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null. This standard deviation is computed from the coordinator activity queuetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. If any values fall into the last histogram bin, the value -1 is returned.
UOW_TOTAL_TIME_TOP	BIGINT	High watermark for unit of work lifetime, in milliseconds. Returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE.

WLM_SET_CLIENT_INFO procedure - Set client information

The WLM_SET_CLIENT_INFO procedure sets client information associated with the current connection at the DB2 server.

By using this procedure, you can set the client's user ID, application name, workstation name, accounting information, or workload information at the DB2 server. Calling this procedure changes the stored values of the relevant transaction processor (TP) monitor client information fields and special register settings for this connection.

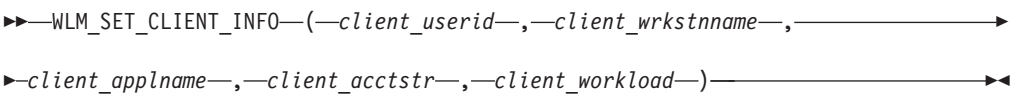
The client information fields are used at the DB2 server for determining the identity of the application or user currently using the connection. The client information fields for a connection are considered during DB2 workload evaluation and also displayed in any DB2 audit records or application snapshots generated for this connection.

Unlike the sqleseti API, this procedure does not set client information at the client but instead sets the corresponding client attributes on the DB2 server. Therefore, you cannot use the sqleqry API to query the client information that is set at the DB2 server using this procedure.

The data values provided with the procedure are converted to the appropriate database code page before being stored in the related TP monitor fields or special registers. Any data value which exceeds the maximum supported size after conversion to the database code page is truncated before being stored at the server. The truncated values are returned by both the TP monitor fields and the special registers when those stored values are queried.

The WLM_SET_CLIENT_INFO procedure is not under transaction control, and client information changes made by the procedure are independent of committing or rolling back units of work. However, because workload reevaluation occurs at the beginning of the next unit of work for each application, you must issue either a COMMIT or a ROLLBACK statement to make client information changes effective.

Syntax



The schema is SYSPROC.

Procedure parameters

client_userid
An input argument of type VARCHAR(255) that specifies the user ID for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the user ID for the client is reset to the default value, which is blank.

client_wrkstnname
An input argument of type VARCHAR(255) that specifies the workstation name for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the workstation name for the client is reset to the default value, which is blank.

client_applname
An input argument of type VARCHAR(255) that specifies the application name for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the application name for the client is reset to the default value, which is blank.

client_acctstr
An input argument of type VARCHAR(255) that specifies the accounting string for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the accounting string for the client is reset to the default value, which is blank.

client_workload
An input argument of type VARCHAR(255) that specifies the workload assignment mode for the client. If you specify NULL, the value remains unchanged. The values are as follows:

SYSDEFAULTADMWORKLOAD
Specifies that the database connection will be assigned to SYSDEFAULTADMWORKLOAD, enabling users with ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority to bypass the normal workload evaluation.

AUTOMATIC
Specifies that the database connection will be assigned to a workload chosen by the workload evaluation that is performed automatically by the server.

Note: The *client_workload* argument is case sensitive.

Authorization

EXECUTE privilege on the WLM_SET_CLIENT_INFO procedure.

Examples

The following procedure call sets the user ID, workstation name, application name, accounting string, and workload assignment mode for the client:

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('db2user', 'machine.torolab.ibm.com',
   'auditor', 'Accounting department', 'AUTOMATIC')
```

The following procedure call sets the user ID to db2user2 for the client without setting the other client attributes:

```
CALL SYSPROC.WLM_SET_CLIENT_INFO('db2user2', NULL, NULL, NULL, NULL)
```

The following procedure call resets the user ID for the client to blank without modifying the values of the other client attributes:

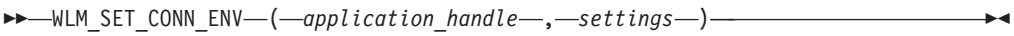
```
CALL SYSPROC.WLM_SET_CLIENT_INFO('', NULL, NULL, NULL, NULL)
```

WLM_SET_CONN_ENV - Enable collection of activity data and measurement of section actuals

The WLM_SET_CONN_ENV procedure enables for a particular connection the collection of activity data and measurement of section actuals (runtime statistics measured during section execution).

Once applied, the settings made by the WLM_SET_CONN_ENV procedure continue to apply until explicitly overwritten by another call to the WLM_SET_CONN_ENV procedure, or until the connection is closed. After the connection is closed, any new connection that reuses the same application handle does not inherit the settings of the previous connection to use that application handle.

Note: If your database was created in Version 9.7 prior to Fix Pack 2, to run this routine you must have already run the db2updv97 command. If your database was created before Version 9.7, it is not necessary to run the db2updv97 command (because the catalog update is automatically taken care of by the database upgrade). If you downgrade to Version 9.7, this routine will no longer work.



The schema is SYSPROC.

Authorization

EXECUTE privilege on the WLM_SET_CONN_ENV procedure.

Parameters

application_handle
An input argument of type BIGINT that specifies the application handle whose connection environment is to be modified. The application handle specified must refer to an existing application (otherwise, SQLSTATE 5U002 is returned). You can use a value of NULL to indicate that the connection whose environment is to be changed is the connection on which the procedure was invoked.

settings

An input argument of type CLOB(8K) that enables you to specify one or more monitor settings. Settings are specified as name value pairs using the format:

<setting name tag>value</setting name tag>

Each setting can be specified a maximum of one time. Setting names are case sensitive. A change to a setting takes effect on the next statement executed; it has no effect on statements already in progress.

The available setting name tags are as follows.

- '<collectactdata>*value*</collectactdata>'

  Specifies what activity data should be collected by the activity event monitor. The possible values are (variations in spaces between words are supported):

Value	Description
NONE	Activity data should not be collected
WITHOUT DETAILS	Data about each activity is sent to any active activities event monitor when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.
WITH DETAILS	Statement and compilation environment data is sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.
WITH DETAILS, SECTION	Statement, compilation environment, section environment data, and section actuals are sent to any active activities event monitor, for those activities that have them. For section actuals to be collected, either *collectsectionactuals* must be set to BASE or the **section_actuals** database configuration parameter must be set to BASE. Section actuals are collected on any partition where the activity data is collected.

Value	Description
WITH DETAILS, SECTION AND VALUES	Statement, compilation environment, section environment data, section actuals, and input data values are sent to any active activities event monitor, for those activities that have them. For section actuals to be collected, either *collectsectionactuals* must be set to BASE or the **section_actuals** database configuration parameter must be set to BASE. Section actuals are collected on any partition where the activity data is collected.
WITH DETAILS AND VALUES	Statement, compilation environment, and input data values are sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

- '<collectactpartition>COORDINATOR</collectactpartition>' or '<collectactpartition>ALL</collectactpartition>'

  Specifies where activity data is collected, either just at the coordinator partition or at all partitions. If *collectactpartition* is not specified, the connection maintains its previous value for *collectactpartition* which by default is COORDINATOR.
- '<collectsectionactuals>NONE</collectsectionactuals>' or '<collectsectionactuals>BASE</collectsectionactuals>'

  Section actuals are collected if *collectsectionactuals* is set to BASE.

Example

The following examples both enable activity collection, without details, on the coordinator partition for the current connection:

```
CALL WLM_SET_CONN_ENV(NULL,'<collectactdata>WITHOUT DETAILS</collectactdata>')

CALL WLM_SET_CONN_ENV(NULL,'<collectactdata>WITHOUT
   DETAILS</collectactdata><collectactpartition>COORDINATOR
   </collectactpartition>')
```

The next example enables collection of activity data with section environment data and section actuals , but no data values, on all partitions for the current connection:

```
CALL WLM_SET_CONN_ENV(NULL, '<collectactdata>WITH DETAILS, SECTION
   </collectactdata><collectactpartition>ALL</collectactpartition>')
```

The following example disables collection of activity data for the current connection.

```
CALL WLM_SET_CONN_ENV(NULL, '<collectactdata>NONE</collectactdata>')
```

Usage notes

The *collectactdata* setting only controls activity data collection at the connection level . An activity might have multiple activity data collection controls applied to it, for example, the connection might be mapped to a service class where the

COLLECT ACTIVITY DATA clause has been applied. In a situation where multiple activity data collection controls are applied, the effective setting is the combination of all the settings. For example:

1. The connection level control is activity data without details.
2. The workload control is none.
3. The service class control is activity data with details and values.
4. When the activity completes execution, detailed information about the activity plus data values is sent to any active event monitors.

If a setting is not specified in the input of the WLM_SET_CONN_ENV procedure, it is not altered in the connection environment.

The effective setting for the collection of section actuals is the combination of the *collectsectionactuals* setting and the **section_actuals** database configuration parameter. For example, if *collectsectionactuals* is set to BASE and the **section_actuals** database configuration parameter value is NONE, the effective setting for the collection of section actuals is BASE (and vice versa). Do not use automatic statistics profiling (enabled using the **auto_stats_prof** database configuration parameter) if *collectsectionactuals* is set to BASE (otherwise, the warning SQLSTATE 01HN2 is returned).

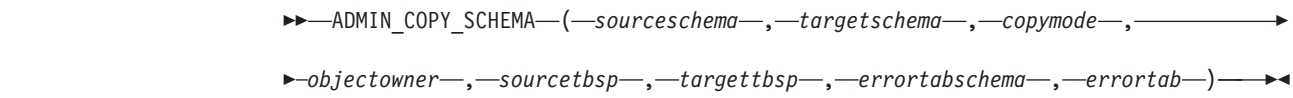
Automatic client rerouting cannot be performed for a connection when activity data and section actuals are being collected (when *collectactdata* is set to any value other than NONE).

Chapter 21. Miscellaneous routines and views

ADMIN_COPY_SCHEMA procedure - Copy a specific schema and its objects

The ADMIN_COPY_SCHEMA procedure is used to copy a specific schema and all objects contained in it. The new target schema objects will be created using the same object names as the objects in the source schema, but with the target schema qualifier. The ADMIN_COPY_SCHEMA procedure can be used to copy tables with or without the data of the original tables.

Syntax



The schema is SYSPROC.

Procedure parameters

sourceschema
An input argument of type VARCHAR(128) that specifies the name of the schema whose objects are being copied. The name is case-sensitive.

targetschema
An input argument of type VARCHAR(128) that specifies a unique schema name to create the copied objects into. The name is case-sensitive. If the schema name already exists, the procedure call will fail and return a message indicating that the schema must be removed prior to invoking the procedure.

copymode
An input argument of type VARCHAR(128) that specifies the mode of copy operation. Valid options are:

- 'DDL': create empty copies of all supported objects from the source schema.
- 'COPY': create empty copies of all objects from the source schema, then load each target schema table with data. Load is done in 'NONRECOVERABLE' mode. A backup must be taken after calling the ADMIN_COPY_SCHEMA, otherwise the copied tables will be inaccessible following recovery.
- 'COPYNO': create empty copies of all objects from the source schema, then load each target schema table with data. Load is done in 'COPYNO' mode.

Note: If *copymode* is 'COPY' or 'COPYNO', a fully qualified filename, for example 'COPYNO /home/mckeough/loadoutput', can be specified along with the *copymode* parameter value. When a path is passed in, load messages will be logged to the file indicated. The file name must be writable by the user ID used for fenced routine invocations on the instance. If no path is specified, then load message files will be discarded (default behavior).

objectowner
An input argument of type VARCHAR(128) that specifies the authorization ID

to be used as the owner of the copied objects. If NULL, then the owner will be the authorization ID of the user performing the copy operation.

sourcetbsp
An input argument of type CLOB(2 M) that specifies a list of source table spaces for the copy, separated by commas. Delimited table space names are supported. For each table being created, any table space found in this list, and the tables definition, will be converted to the nth entry in the *targettbsp* list. If NULL is specified for this parameter, new objects will be created using the same table spaces as the source objects use.

targettbsp
An input argument of type CLOB(2 M) that specifies a list of target table spaces for the copy, separated by commas. Delimited table space names are supported. One table space must be specified for each entry in the *sourcetbsp* list of table spaces. The nth table space in the *sourcetbsp* list will be mapped to the nth table space in the *targettbsp* list during DDL replay. It is possible to specify 'SYS_ANY' as the final table space (an additional table space name, that does not correspond to any name in the source list). When 'SYS_ANY' is encountered, the default table space selection algorithm will be used when creating objects (refer to the IN *tablespace-name1* option of the CREATE TABLE statement documentation for further information on the selection algorithm). If NULL is specified for this parameter, new objects will be created using the same table spaces as the source objects use.

errortabschema
An input and output argument of type VARCHAR(128) that specifies the schema name of a table containing error information for objects that could not be copied. This table is created for the user by the ADMIN_COPY_SCHEMA procedure in the SYSTOOLSPACE table space. If no errors occurred, then this parameter is NULL on output.

errortab
An input and output argument of type VARCHAR(128) that specifies the name of a table containing error information for objects that could not be copied. This table is created for the user by the ADMIN_COPY_SCHEMA procedure in the SYSTOOLSPACE table space. This table is owned by the user ID that invoked the procedure. If no errors occurred, then this parameter is NULL on output. If the table cannot be created or already exists, the procedure operation fails and an error message is returned. The table must be cleaned up by the user following any call to the ADMIN_COPY_SCHEMA procedure; that is, the table must be dropped in order to reclaim the space it is consuming in SYSTOOLSPACE.

Table 249. ADMIN_COPY_SCHEMA errortab format

Column name	Data type	Description
OBJECT_SCHEMA	VARCHAR(128)	Schema name of the object for which the copy command failed.
OBJECT_NAME	VARCHAR(128)	Name of the object for which the copy command failed.
OBJECT_TYPE	VARCHAR(30)	Type of object.
SQLCODE	INTEGER	The error SQLCODE.
SQLSTATE	CHAR(5)	The error SQLSTATE.
ERROR_TIMESTAMP	TIMESTAMP	Time of failure for the operation that failed.

Table 249. ADMIN_COPY_SCHEMA errortab format (continued)

Column name	Data type	Description
STATEMENT	CLOB(2 M)	DDL for the failing object. If the failure occurred when data was being loaded into a target table, this field contains text corresponding to the load command that failed.
DIAGTEXT	CLOB(2 K)	Error message text for the failed operation.

Authorization

In order for the schema copy to be successful, the user must have the CREATE_SCHEMA privilege as well as DB2 object-specific privileges.

Example: CREATE_TABLE privilege is needed to copy a table and CREATE_INDEX privilege is needed to copy an index under the ADMIN_COPY_SCHEMA command.

If a table in the source schema is protected by label based access control (LBAC), the user ID must have LBAC credentials that allow creating that same protection on the target table. If copying with data, the user ID must also have LBAC credentials that allow both reading the data from the source table and writing that data to the target table.

EXECUTE privilege on the ADMIN_COPY_SCHEMA procedure is also needed.

Example

```
CALL SYSPROC.ADMIN_COPY_SCHEMA('SOURCE_SCHEMA', 'TARGET_SCHEMA',
   'COPY', NULL, 'SOURCETS1 , SOURCETS2', 'TARGETTS1, TARGETTS2,
   SYS_ANY', 'ERRORSCHEMA', 'ERRORNAME')
```

Restrictions

- Only DDL *copymode* is supported for HADR databases.
- XML with COPY or COPY NO is not supported.
- Using the ADMIN_COPY_SCHEMA procedure with the COPYNO option places the table spaces in which the target database object resides in backup pending state. After the load operation completes, target schema tables are in set integrity pending state, and the ADMIN_COPY_SCHEMA procedure issues a SET INTEGRITY statement to get the tables out of this state. Because the table spaces are already in backup pending state, the SET INTEGRITY statement fails. For information on how to resolve this problem, see “Copying a schema”.

Usage notes

- References to fully qualified objects within the objects being copied will not be modified. The ADMIN_COPY_SCHEMA procedure only changes the qualifying schema of the object being created, not any schema names that appear within SQL expressions for those objects. This includes objects such as generated columns and trigger bodies.
- This procedure does not support copying the following objects:
  - index extensions

  - nicknames
  - packages
  - typed tables
  - array types
  - user-defined structured types (and their transform functions)
  - typed views
  - jars (Java routine archives)
  - staging tables
  - aliases with base objects that do not belong to the same source schema
- If one of the above objects exists in the schema being copied, the object is not copied but an entry is added to the error table indicating that the object has not been copied.
- When a replicated table is copied, the new copy of the table does not have subscriptions enabled. The table is recreated as a basic table only.
- The operation of this procedure requires the existence of the SYSTOOLSPACE table space. This table space is used to hold metadata used by the ADMIN_COPY_SCHEMA procedure as well as error tables returned by this procedure. If the table space does not exist, an error is returned.
- Statistics for the objects in the target schema are set to default.
- If a table has a generated identity column, and *copymode* is either 'COPY' or 'COPYNO', the data values from the source table are preserved during the load.
- A new catalog entry is created for each external routine, referencing the binary of the original source routine.
- If a table is in set integrity pending state at the beginning of the copy operation, the data is not loaded into the target table and an entry is logged in *errortab* indicating that the data was not loaded for that table.
- If a Load or DDL operation fails, an entry is logged in *errortab* for any object that was not created. All objects that are successfully created remain. To recover, a manual load can be initiated, or the new schema can be dropped using the ADMIN_DROP_SCHEMA procedure and the ADMIN_COPY_SCHEMA procedure can be called again.
- During DDL replay, the default schema is overridden to the target schema if it matches the source schema.
- The function path used to compile a trigger, view or SQL function is the path used to create the source object, with the following exception: if the object's function path contains the source schema name, this entry in the path is modified to the target schema name during DDL replay.
- Running multiple ADMIN_COPY_SCHEMA procedures will result in deadlocks. Only one ADMIN_COPY_SCHEMA procedure call should be issued at a time. Changes to tables in the source schema during copy processing might mean that the data in the target schema is not identical following a copy operation.
- Careful consideration should be taken when copying a schema with tables from a table space in a single-partition database partition group to a table space in a multiple-partition database partition group. Unless automatic distribution key selection is preferred, the distribution key should be defined on the tables before the copy schema operation is undertaken. Altering the distribution key can only be done to a table whose table space is associated with a single-partition database partition group.

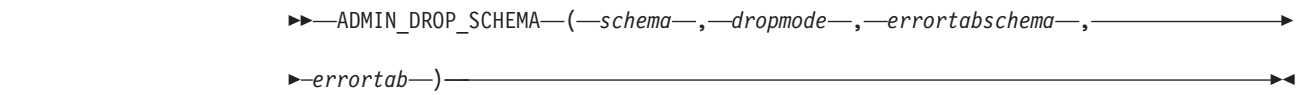
Transactional considerations

- If the ADMIN_COPY_SCHEMA procedure is forced to rollback due to a deadlock or lock timeout during its processing, any work performed in the unit of work that called the ADMIN_COPY_SCHEMA procedure is also rolled back.
- If a failure occurs during the DDL phase of the copy, all the changes that were made to the target schema are rolled back to a savepoint.
- If *copymode* is set to 'COPY' or 'COPYNO', the ADMIN_COPY_SCHEMA procedure commits once the DDL phase of the copy is complete, also committing any work done in the unit of work that called the procedure.

ADMIN_DROP_SCHEMA procedure - Drop a specific schema and its objects

The ADMIN_DROP_SCHEMA procedure is used to drop a specific schema and all objects contained in it.

Syntax



The schema is SYSPROC.

Procedure parameters

schema
An input argument of type VARCHAR(128) that specifies the name of the schema being dropped. The name must be specified in uppercase characters.

dropmode
Reserved for future use and should be set to NULL.

errortabschema
An input and output argument of type VARCHAR(128) that specifies the schema name of a table containing error information for objects that could not be dropped. The name is case-sensitive. This table is created for the user by the ADMIN_DROP_SCHEMA procedure in the SYSTOOLSPACE table space. If no errors occurred, then this parameter is NULL on output.

errortab
An input and output argument of type VARCHAR(128) that specifies the name of a table containing error information for objects that could not be dropped. The name is case-sensitive. This table is created for the user by the ADMIN_DROP_SCHEMA procedure in the SYSTOOLSPACE table space. This table is owned by the user ID that invoked the procedure. If no errors occurred, then this parameter is NULL on output. If the table cannot be created or already exists, the procedure operation fails and an error message is returned. The table must be cleaned up by the user following any call to ADMIN_DROP_SCHEMA; that is, the table must be dropped in order to reclaim the space it is consuming in SYSTOOLSPACE.

Table 250. ADMIN_DROP_SCHEMA errortab format

Column name	Data type	Description
OBJECT_SCHEMA	VARCHAR(128)	Schema name of the object for which the drop command failed.
OBJECT_NAME	VARCHAR(128)	Name of the object for which the drop command failed.
OBJECT_TYPE	VARCHAR(30)	Type of object.
SQLCODE	INTEGER	The error SQLCODE.
SQLSTATE	CHAR(5)	The error SQLSTATE.
ERROR_TIMESTAMP	TIMESTAMP	Time that the drop command failed.
STATEMENT	CLOB(2 M)	DDL for the failing object.
DIAGTEXT	CLOB(2 K)	Error message text for the failed drop command.

Authorization

Drop authority is needed on all objects being removed for the user calling this procedure.

EXECUTE privilege on the ADMIN_DROP_SCHEMA procedure is also needed.

Example

```
CALL SYSPROC.ADMIN_DROP_SCHEMA('SCHNAME', NULL, 'ERRORSCHEMA', 'ERRORTABLE')
```

The following is an example of output for this procedure.

```
Value of output parameters
--------------------------
Parameter Name : ERRORTABSCHEMA
Parameter Value : ERRORSCHEMA <-- error!

Parameter Name : ERRORTAB
Parameter Value : ERRORTABLE <-- error!

Return Status = 0
```

The return status is not zero only when an internal error has been detected (for example, if SYSTOOLSPACE does not exist).

Errors can be checked by querying the error table:

```
SELECT * FROM ERRORSCHEMA.ERRORTABLE
```

Usage notes

- If objects in another schema depend on an object being dropped, the default DROP statement semantics apply.
- This procedure does not support dropping the following objects:
  - index extensions
  - nicknames
  - packages
  - typed tables

  - array types
  - user-defined structured types (and their transform functions)
  - typed views
  - jars (Java routine archives)
  - staging tables
- If one of the above objects exists in the schema being dropped, neither the object nor the schema is dropped, and an entry is added to the error table indicating that the object was not dropped.
- The operation of this procedure requires the existence of the SYSTOOLSPACE table space. This table space is used to hold metadata used by the ADMIN_DROP_SCHEMA procedure as well as error tables returned by this procedure. If the table space does not exist, an error is returned.

ADMIN_MOVE_TABLE procedure - Move an online table

The ADMIN_MOVE_TABLE stored procedure moves the data in an active table into a new table object with the same name, while the data remains online and available for access. This stored procedure creates a protocol table composed of rows containing status information and configuration options related to the table to be moved. The return set from this procedure are the rows from that protocol table related to the table to be moved.

This stored procedure uses the following terminology:

Source table
: The original table name that is passed in as a parameter into the stored procedure. This is the table to be moved.

Target table
: A table created by the stored procedure using the table definition passed in through the stored procedure. All of the data from the source table is copied into this table and then it is renamed to the same name as the source table.

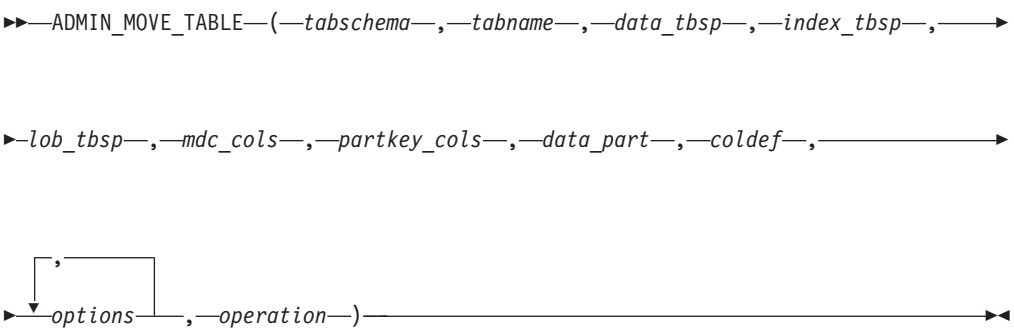
Staging table
: A table created by the stored procedure. The staging table stores any update, delete or insert changes that occur on the source table during the execution of the table move. This table is dropped when the move is complete.

Syntax

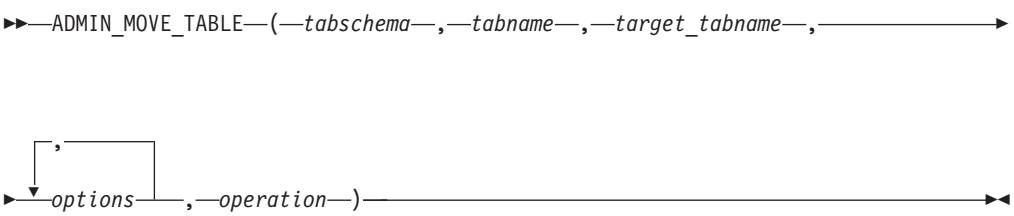
There are two equally valid methods to invoke ADMIN_MOVE_TABLE. The first method allows you to modify only certain parts of the table definition for the target table. For instance, if you had a table definition that is quite large (several KB), and all you want to do Is modify the table spaces for the table, you can do so without having to determine the entire CREATE TABLE statement needed to recreate the source table. All you need to do is to fill out the data_tbsp, index_tbsp, and lob_tbsp parameters, leaving the other optional parameters blank.

The second method provides you with more control and flexibility by allowing you to create the target table beforehand, rather than having the stored procedure create the target table. This enables you to create a target table that would not be possible using the first method.

Method 1:



Method 2:



The schema for both methods is SYSPROC.

Procedure parameters

tabschema
This input parameter specifies the name of the schema which contains the table to be moved. This parameter is case sensitive and has a data type of VARCHAR(128).

tabname
This input parameter specifies the name of the table to be moved. This parameter is case sensitive and has a data type of VARCHAR(128)

data_tbsp
This input parameter specifies the new data table space for the target table. If a value is provided, the *index_tbsp* and *lob_tbsp* parameters are required. If a value is not provided, the data table space of the source table is used. This parameter is case sensitive and has a data type of VARCHAR(128). This parameter can be NULL or the empty string.

index_tbsp
This input parameter specifies the new index table space for the target table. If a value is provided, the *data_tbsp* and *lob_tbsp* parameters are required. If a value is not provided, the index table space of the source table is used. This parameter is case sensitive and has a data type of VARCHAR(128). This parameter can be NULL or the empty string.

lob_tbsp
This input parameter specifies the new LOB table space for the target table. If a value is provided, the *data_tbsp* and *index_tbsp* parameters are required. If a value is not provided, the LOB table space of the source table is used. This parameter is case sensitive and has a data type of VARCHAR(128). This parameter can be NULL or the empty string.

mdc_cols
This input parameter provides the multi-dimensional column (MDC)

specification for the target table. The values are entered as a comma separated list of the columns used to cluster data in the target table along multiple dimensions. If a value of NULL or "-" is given, the ORGANIZE BY DIMENSIONS clause is not used. If an empty string or a single blank is given, the procedure checks whether there is an MDC specification on the source table, and uses that specification if located. This parameter has a data type of VARCHAR(32672) and has the same format as the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. This parameter can be NULL, the empty string, or a single blank.

Example: 'C1, C4, (C3,C1), C2'

partkey_cols
This input parameter provides the partitioning key columns specification for the target table. The values are entered as a comma separated list of the key columns that specify how the data is distributed across multiple database partitions. If a value of NULL or "-" is given, the PARTITIONING KEY clause is not used. If an empty string or a single blank is given, the procedure checks whether there is a partitioning key columns specification on the source table, and uses that specification if located. This parameter has a data type of VARCHAR(32672) and has the same format as the DISTRIBUTE BY HASH clause of the CREATE TABLE statement.

Example: 'C1, C3'

data_part
This input parameter provides the data partitioning specification for the target table. This statement defines how to divide table data across multiple storage objects (called data partitions), according to the values in one or more of the table columns. If a value of NULL or "-" is given, the PARTITION BY RANGE clause is not used. If an empty string or a single blank is given, the procedure checks whether there is a data partition scheme on the source table, and uses that information (including partition name) if located. This parameter has a data type of VARCHAR(32672) and has the same format as the PARTITION BY RANGE clause of the CREATE TABLE statement.

Example: '(C1) (STARTING FROM (1) EXCLUSIVE ENDING AT (1000) EVERY (100))'

coldef
This input parameter specifies a new column definition for the target table, allowing you to change the column types as long as they are compatible; however, the column names must remain the same.

This also provides the ability to add new columns and drop existing columns. When adding a column, it must be defined as either nullable or have a default value set. Also, a column can only be dropped if there is a unique or primary index on the table and the column to be dropped is not a part of that unique or primary index. This parameter has a data type of VARCHAR(32672). This parameter can be NULL or the empty string.

Example: 'C1 INT, C2 INT DEFAULT 0'

target_tabname
This input parameter provides the name of an existing table to use as the target table during the move. The following changes can be made to the target table being passed in:

- The data, index and LOB table spaces can be changed
- The multi dimensional column (MDC) specification can be added or changed

- The partitioning key columns specification can be added or changed
- The data partitioning specification can be added or changed
- Data compression can be added or removed
- A new column definition can be specified; however the same restrictions as when specifying the *coldef* parameter apply here.

The following restrictions apply to the named table:

- The table must exist in the same schema as the source table
- The table must be empty
- No typed tables, materialized query tables (MQT), staging tables, remote tables or clustered tables are permitted

If this parameter is set to NULL or the empty string, the stored procedure uses the same definition as the source table. This parameter is case sensitive and has a data type of VARCHAR(128).

options

This set of comma separated input parameters defines any options used by the stored procedure.

- KEEP: This option keeps a copy of the original source table under a different name. If the source table name is T1, then after the move that table will be automatically renamed to something such as T1AAAAVxo. You can retrieve the exact name of the source table in the returned protocol table, under the ORIGINAL key. You may set this option at any point up to and including the SWAP phase.
- COPY_USE_LOAD: This option uses the non-recoverable db2Load API to copy the data from the source table to the target table. You may set this option at any point up to and including the COPY phase. In releases lower than DB2 Version 9.7 Fix Pack 2, the FORCE option must be specified if the COPY_USE_LOAD is used.
- COPY_WITH_INDEXES: This option creates indexes before copying the source table; however, the default is to create the indexes after copying the source table. The advantages of this option are that index creation after copying requires a whole table scan per index and that the index creation is a transaction that requires active log space. If the LOGINDEXREBUILD database configuration parameter is on, significant log space is required for building the indexes in a short time frame. One disadvantage of this option is that copy performance is reduced because indexes need to be maintained on the target table. Also, the resulting indexes many contain "pseudo" deleted keys, and the indexes are not as well balanced as if the indexes were created after the copy. You may set the COPY_WITH_INDEXES option at any point up to and including the COPY phase.
- FORCE: If the force option is set, the SWAP phase does not check to see if the source table has changed its table definition. In releases lower than DB2 Version 9.7 Fix Pack 2, the FORCE option must be specified if the COPY_USE_LOAD is used. You may set this option at any point up to and including the SWAP phase.
- NO_STATS: This option does not start RUNSTATS or any statistic copying on the target table. If you use the AUTO_RUNSTATS or AUTO_STMT_STATS database configuration parameters, DB2 will automatically create new statistics afterwards. For backwards compatibility, STATS_NO is also accepted. You may set the NO_STATS option at any point up to and including the SWAP phase.

- COPY_STATS: This option copies the statistics from the source table to the target table before performing the swap. This may cause inaccurate physical statistics, especially if the page size is changed. However, setting this option saves computing time as RUNSTATS is not called to compute new statistics. Also, the optimizer may choose the same access plans, because the statistics are the same. For backwards compatibility, STATS_COPY is also accepted. You may set the STATS_COPY option at any point up to and including the SWAP phase.
- NO_AUTO_REVAL: This option prevents automatic revalidation on the table, and instead, recreates all triggers and views. The NO_AUTO_REVAL option can be set only in the INIT phase.
- REORG: This option sets up an extra offline REORG on the target table before performing the swap. If you use this option to improve your compression dictionary, be advised that using the default sampling approach is a better method to create an optimal compression dictionary. However, if you require an optimal XML compression dictionary, then REORG is the only method. You may set the REORG option at any point up to and including the SWAP phase.
- NO_TARGET_LOCKSIZE_TABLE: This option does not keep locksize table on the target table during the COPY and SWAP phases. The default is to have locksize table on the target table to prevent locking overhead, when no unique index is specified on the source table. This option is available starting in Version 9.7 Fix Pack 1 and later fix packs.
- CLUSTER: This option reads the data from the source table with an ORDER BY clause when a cluster index exists on the source table or a copy index has been specified. This option is available starting in Version 9.7 Fix Pack 1 and later fix packs.
- NON_CLUSTER: This options reads the data from the source table without an ORDER BY clause regardless if a cluster index or copy index has been specified. Note: When neither CLUSTER or NON_CLUSTER options are specified, it will read the data from the source table with an ORDER BY clause only when a cluster index exists on the source table. This option is available starting in Version 9.7 Fix Pack 1 and later fix packs.
- LOAD_MSGPATH <path>: This option can be used to define the load message file path when the COPY_USE_LOAD option specified. If the LOAD_MSGPATH option is not specified, then diagpath will be used as the default path. This option is available starting in DB2 Version 9.7 Fix Pack 2.

This list of options is not case sensitive and has a data type of VARCHAR(128). The list value can be NULL or the empty string.

operation

This input parameter specifies which operation the stored procedure is to execute. There are two ways of calling the stored procedure: using the MOVE command to execute all the operations at one time; or by using the individual commands to execute the table move one step at a time. The main advantage of this second method is that you control when the SWAP phase actually occurs, thereby determining when the table is briefly taken offline. This allows you to make the move during a period of low system activity. If you use the individual commands, they must be called in the following order: INIT, COPY, REPLAY, VERIFY (optional), and SWAP.

- MOVE: Performs the entire table move (INIT, COPY, REPLAY, and SWAP operations) in one step.

- INIT: Verifies that a table move can take place, and initializes all of the data needed during the table move process (the target table, staging table, and the triggers on the source table).
- COPY: Copies the content from the source table to the target table. Any updates, deletes, or inserts occurring on the source table during this time are captured and stored in the staging table. New indexes are created at the end of the COPY phase, unless the COPY_WITH_INDEXES option is selected. Also, if needed, secondary indexes are created on the source and target tables to improve performance during the REPLAY phase. COPY can be used only after the INIT phase has completed.
- REPLAY: Copies into the target table any rows that have changed in the source table since the COPY phase began. REPLAY can be used only after the COPY phase has completed.
- VERIFY: Optionally checks if the table contents of the source and target tables are identical. This process involves obtaining a shared lock on the source and target tables, replaying any changes that have occurred on the source table, and then performing a comparison. If the table has a unique index, this command compares all values between columns that are in both tables. Otherwise, this command compares all values between columns that are in both tables (except for LONG, LOB or XML columns). This is an expensive operation and caution should be taken to decide if it is useful for your move. VERIFY can be used only after the COPY or REPLAY phases have completed.
- SWAP: Executes the REPLAY phase until the number of changes applied during the last scan of the staging table is less than the REPLAY_THRESHOLD value stored in the protocol table. The source table is then taken offline briefly to finish the final REPLAY, and then this command swaps the source table with target table and brings the table back online. SWAP can be used after the COPY phase has completed, but ideally after the REPLAY phase has been called.
- CLEANUP: Drops the staging table, any non-unique indexes or triggers created on the source table by the stored procedure, and the source table if the KEEP option has not been set. CLEANUP can be called if the command failed during the SWAP phase.
- CANCEL: Cancels a multi-step table move while between phases, or cancels a failed table move operation. Executing this command requires that the operation status is not in COMPLETED or CLEANUP state. CANCEL clears up all intermediate data (the indexes, the staging table, the target table, and the triggers on the source table).

This parameter is not case sensitive and has a data type of VARCHAR(128).

Authorization

You must have either SQLADM or DBADM authority to invoke the ADMIN_MOVE_TABLE stored procedure. You must also have the appropriate object creation authorities, including authorities to issue the SELECT statement on the source table, and to issue the INSERT statement on the target table.

Examples

This example calls the stored procedure using the first method, where the target table is defined within the procedure, to move a table named T1 which is located in the schema "SVALENTI".

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'ACCOUNTING',
'ACCOUNT_IDX',
'ACCOUNT_LONG',
'',
'',
'',
'CUSTOMER VARCHAR(80), REGION CHAR(5), YEAR INTEGER, CONTENTS CLOB',
'',
'MOVE')
```

The following is an example of output from this query

```
Result set 1
------------

KEY                          VALUE
-----------------------      -----------------------------------
AUTHID                       SVALENTI
CLEANUP_END                  2009-02-13-11.34.07.609575
CLEANUP_START                2009-02-13-11.34.07.369331
COPY_END                     2009-02-13-11.34.05.148018
COPY_OPTS                    BY_KEY,OVER_INDEX
COPY_START                   2009-02-13-11.34.04.841292
COPY_TOTAL_ROWS              100
INDEXNAME                    T1_INDEX
INDEXSCHEMA                  SVALENTI
INDEX_CREATION_TOTAL_TIME    0
INIT_END                     2009-02-13-11.34.04.552875
INIT_START                   2009-02-13-11.34.03.013563
PAR_COLDEF                   CUSTOMER VARCHAR(80), REGION CHAR(5),
                                 YEAR INTEGER, CONTENTS CLOB
REPLAY_END                   2009-02-13-11.34.06.198369
REPLAY_START                 2009-02-13-11.34.05.164582
REPLAY_TOTAL_ROWS            100
REPLAY_TOTAL_TIME            5
STATUS                       COMPLETE
SWAP_END                     2009-02-12-11.34.07.214447
SWAP_RETRIES                 0
SWAP_START                   2009-02-13-11.34.06.244506
VERSION                      09.07.0000

22 record(s) selected.

Return Status = 0
```

This example calls the stored procedure using the second method , where the target table is created outside the procedure and is then named within the *target_tabname* parameter, to move the same table as above.

The first step is to create the table manually:

```
CREATE TABLE SVALENTI.T1_TARGET (
   CUSTOMER VARCHAR(80),
   REGION CHAR(5),
   YEAR INTEGER,
   CONTENTS CLOB)
IN ACCOUNTING
INDEX IN ACCOUNT_IDX
LONG IN ACCOUNT_LONG'
```

Then call the stored procedure and pass in the name of the target table:

```
CALL SYSPROC.ADMIN_MOVE_TABLE(
'SVALENTI',
'T1',
'T1_TARGET',
'',
'MOVE')
```

The following is an example of output from this query

```
Result set 1
------------

KEY                         VALUE
-------------------------   -----------------------------------------
AUTHID                      SVALENTI
CLEANUP_END                 2009-02-13-11.37.49.283090
CLEANUP_START               2009-02-13-11.37.49.125786
COPY_END                    2009-02-13-11.37.47.806060
COPY_OPTS                   BY_KEY,OVER_INDEX
COPY_START                  2009-02-13-11.37.47.446616
COPY_TOTAL_ROWS             0
INDEXNAME                   T1_INDEX
INDEXSCHEMA                 SVALENTI
INDEX_CREATION_TOTAL_TIME   1
INIT_END                    2009-02-13-11.37.47.287703
INIT_START                  2009-02-13-11.37.46.052952
PAR_COLDEF                  using a supplied target table so COLDEF
                                could be different
REPLAY_END                  2009-02-13-11.37.48.785503
REPLAY_START                2009-02-13-11.37.47.822109
REPLAY_TOTAL_ROWS           0
REPLAY_TOTAL_TIME           0
STATUS                      COMPLETE
SWAP_END                    2009-02-13-11.37.48.977745
SWAP_RETRIES                0
SWAP_START                  2009-02-13-11.37.48.825228
VERSION                     09.07.0000
22 record(s) selected.

Return Status = 0
```

Usage notes

Suggestions for best results when using this procedure:

- Avoid making multiple moves into same table space at the same time. This prevents fragmentation on the target table space.
- Run this procedure when activity on the table is low. Avoid mass data loads or deletes so that parallel read access is not a problem.
- Use a multi-step move operation. The INIT and COPY phases can be called at any time. Execute the REPLAY phase multiple times in order to keep the staging table size small, and then issue the SWAP during a time of low activity on the table.
- Check if offline methods are a better choice for your table move, especially when considering tables without unique indexes and for tables with no index.

Operations that are restricted on the source table

The stored procedure relies on triggers to capture any changes made to the source table. There are some operations that could affect the source table but which do not fire triggers. This could result in inconsistencies between the source and target table that cannot easily be detected by the stored procedures. These operations include:

- TRUNCATE TABLE (without restrict when delete triggers)
- IMPORT ... REPLACE INTO ...
- LOAD TABLE
- ALTER TABLE
- REORG (both online and offline)

These operations will be restricted on the source table using a new table-level state flag. The flag is set during the INIT phase and cleared during the CLEANUP or CANCEL phase. Restricted operations will fail with SQL0668N reason code 10 (sqlstate 57016).

Operations that will affect the table move operation

There are operations that can cause the stored procedure to fail while a move is in progress. These operations include:

- Dropping the SYSTOOLSPACE table space
- Dropping/Renaming the source table
- Dropping/Renaming any of the temporary objects created by OTM in the INIT phase (target table, staging table, triggers on source table, protocol table)
- Altering values in the protocol table that are not listed as user configurable

Naming convention for temporary objects

To avoid naming conflicts when creating temporary objects, the following naming convention is used:

- Postfix
  - "t" for target
  - "s" for staging
  - "o" for original
  - "g" for generated
  - "i" for insert trigger
  - "d" for delete trigger
  - "u" for before update trigger
  - "v" for after update trigger
- Names are built consisting of `<characters from name of object><base64 encoded hash key over name of object><postfix>`.
- If length of name would exceed object length (128 bytes) `<characters from name of object>` gets shorter.
- Hash value gets calculated from the object name and is encoded similar to base64 encoding.

Sample:

```
Name of object: T1
Staging object: T1AAAAVxs
Target object: T1AAAAVxt
Original object: T1AAAAVxo
Generated index: T1AAAAVxg (if table has no index)
Insert trigger: T1AAAAVxi
Delete trigger: T1AAAAVxd
Before update trigger: T1AAAAVxu
After update trigger: T1AAAAVxv
```

Online table move with compression and dictionary creation

There are several methods to create a data compression dictionary using Online Table Move. Compression must either be enabled on the source table or specified to be active in the new table definition if provided.

Create dictionary with sampling is the default method of Dictionary creation through Online Table Move. If compression is turned on for the table, then before performing the COPY operation, a Bernoulli sampling of the data from the source table is inserted into the target table, where the amount of data sampled is specified in the DEEPCOMPRESSION_SAMPLE field in the protocol table. The compression dictionary is then created based off of this random sample, and therefore results in an optimal compression dictionary.

Please note, that an XML compression dictionary will not be created through the sampling method. This is due to the fact that db2Inspect is used to create the compression dictionary, and db2Inspect currently does not have the ability to create an XML compression dictionary. The XML compression dictionary will be created through automatic dictionary creation (ADC).

Create dictionary with automatic dictionary creation (ADC) is the standard method of Dictionary creation with tables in DB2. By simply turning on compression for the table, DB2 will automatically create the dictionary as data is inserted into the table. This will result in a non-optimal compression dictionary. Please note that the DEEPCOMPRESSION_SAMPLE field in the protocol table will have to be set to 0 to avoid having the stored procedure attempt to create a better compression dictionary.

The create dictionary with REORG method of Dictionary creation results in a dictionary being created that reflects any activity on the source table that occurred while the COPY phase was in process. This is done by performing a REORG before the SWAP phase with the RESETDICTIONARY option set. An optimal dictionary will be created, however depending on the size of the table the REORG could take a long time. Also, if an optimal XML dictionary is required, REORG is the only method that will produce one. It is advised to use the sampling method of dictionary creation.

Online table move and statistics on the table

The default behavior when performing a table move on a table where statistics are collected is to perform RUNSTATS on the table during the SWAP phase. If a statistics profile is found, RUNSTATS will be called using the statistics profile. Otherwise, RUNSTATS will be called with the options "WITH DISTRIBUTION ON COLUMNS (...) AND SAMPLE DETAILED INDEXES ALL".

If the COPY_STATS option has been set, the statistics from the source table are copied to the target table before performing the swap. Copying statistics may cause inaccurate physical statistics especially if changing page size. However, it will save on computing time as RUNSTATS does not have to be called to compute new statistics. Also, the optimizer may choose the same access plans, because the statistics are the same (plan stability). The statistics that are copied are in the SYSSTAT.TABLES, SYSSTAT.COLUMNS, SYSSTAT.COLDIST, SYSSTAT.INDEXES, and SYSSTAT.COLGROUPS catalog views.

If the NO_STATS option has been set, the stored procedure does not perform RUNSTATS or any statistic copying on the target table. If you use AUTO_RUNSTATS or AUTO_STMT_STATS, DB2 will automatically create new statistics

Online table move with LOAD used for COPY

When using the COPY_USE_LOAD option, it is necessary to perform a backup of the target table space(s) before the SWAP phase in order to ensure recoverability. A backup can be created by issuing a statement such as the following:

```
BACKUP DB dbname TABLESPACE targetDataTablespace, targetIndexTablespace
    ONLINE TO <destination>
```

In releases lower than DB2 Version 9.7 Fix Pack 2, the FORCE option must be specified if the COPY_USE_LOAD is used. Else, the SWAP phase will not execute and you will receive an error.

Online table move with generated columns

The Table Move stored procedure treats any generated columns in the source table specially. Below is a description of how the different types of generated columns are handled.

A **row change timestamp column** is a column that holds a timestamp representing the time when a row was last changed.

If a row change timestamp column is found in the source table, the values of this column after the Table Move operation is complete will not be the same as they where before the Table Move operation. The values of the column after the Table Move will represent the time at which the rows where inserted/updated in the new table object. This is done because the actual rows are being changed and the row change timestamp column values should therefore reflect these changes.

If a new table definition is supplied, and a column is defined as a row change timestamp column in the source table but not in the new table definition, then the column will not be a row change timestamp column.

An **identity column** is a column that automatically generates a value for the column when a row is inserted into the table.

If an identity column is found in the source table, the values of this column after the Table Move operation is complete will be identical to the values that were present before the Table Move operation. However, there is no way to determine the "last/next" value for the identity column in the source table. Therefore, when creating the identity column on the target table the value generation will be set to begin from the next "uncached" value. This is the same behavior that happens when the database restarts (stop/start). This behavior is documented in the information center, in the "ALTER TABLE" entry, under the "SET NO CACHE or CACHE integer-constant" heading of the "identity-alteration" section which can be found here.

The column will initially be created as a regular column in the target table, and then be altered to be an identity column during the brief offline period of the SWAP phase. This is done because the column may have been created as "GENERATED ALWAYS", and that would block the stored procedure from being able to insert the exact values from the source table into the column in the target table.

If a new table definition is specified, and a column is specified to be an identity column in the new table definition, then the stored procedure will check to see if the definition of the identity column matches the definition of the column in the

source table. If they are a match, the stored procedure will continue as previously described. If they are not a match, the stored procedure will use the new identity column definition. Please note that this will restart the identity column counter with whatever the start value is specified as, however the current values of the rows in the column will remain the same.

If a new table definition is specified, and a column that is specified as an identity column in the source table is not specified as an identity column in the new table definition, then the stored procedure will still create the column as an identity column in the target table using the same specification found in the source table. This is done so that users do not need to look up the definition of the existing identity column and re-enter it into the new table definition. If the user does not want to keep the column as an Identity column, then they can alter the target table after the call to the stored procedure to remove the identity specification from the column.

An **expression column** is a column that automatically generates a value for the column based on an expression when a row is inserted into the table.

If an expression column is found in the source table, the values of this column after the Table move operation is complete will be identical to the values that were present before the Table Move operation.

The column will originally be created as a regular column in the target table, and then be altered to be an expression column during the brief offline period of the SWAP phase. This is done because expression columns are created as "GENERATED ALWAYS", and do not allow inserts into that column. However, In order to alter the column in the target table to be an expression column, set integrity will briefly be turned off on the target table. The ALTER statement is performed, and then integrity is set back on with the "GENERATED COLUMN IMMEDIATE UNCHECKED" option.

The stored procedure will not support column expressions that include the table name (i.e. table 'T1' with expression (T1.C *5)) in either the source table or the target table. To remedy this, the user can alter the column to change the expression to not include the table name.

If a new table definition is specified, and a column is specified to be an expression column in the new table definition, then the stored procedure will check to see if the definition of the expression column matches the definition of the column in the source table by performing a basic string to string comparison. If they are a match, the stored procedure will continue as previously described. If they are not a match, the stored procedure will use the new expression column definition. Please note that the current values of the rows in the column will remain the same.

If a new table definition is specified, and a column that is specified as an expression column in the source table is not specified as an expression column in the new table definition, then the stored procedure will still create the column as an expression column in the target table using the same specification found in the source table. This is done so that users do not need to look up the definition of the existing expression column and re-enter it into the new table definition. If the user does not want to keep the column as an Expression column, then they can alter the target table after the call to the stored procedure to remove the Expression Specification from the column.

Online table move and objects and privileges that are preserved

The stored procedure will preserve the following objects when a Table Move is performed:

Views During the brief offline period during the SWAP phase, the views are dropped from the source table and are recreated on the target table.

Transfer of ownership is also performed to change the ownership of the view back to the original owner.

Triggers

During the brief offline period during the SWAP phase, the triggers are dropped from the source table and are recreated on the target table.

Transfer of ownership is also performed to change the ownership of the trigger back to the original owner.

Indexes

Indexes are created onto the target table at several times during the table move procedure. Indexes are first created at the end of the COPY phase, unless the COPY_WITH_INDEXES option is set then the indexes will first be created at the beginning of the COPY phase. The store procedure will then also look for any newly created indexes, judging by index name alone, at the beginning of the REPLAY and SWAP phases. If new indexes are found, they will be created. However, the stored procedure will not look to see if any indexes have been deleted on the source table.

The index names will be the same as they were on the source table for user created indexes. However, system created indexes can not be guaranteed to have the same name.

The indexes that will be preserved are of the following type: 'REG','CLUST', and 'XVIL'.

Any user created indexes that reference a column that is being dropped in the target table will not be preserved.

When moving from a source partitioned table to a target partitioned table, the partitioned attribute of the index will be preserved. When moving from a source partitioned table to a target non-partitioned table, or vice-versa, the partitioned attribute will be decided by the default behavior of the database.

Constraints

Constraints (other than referential constraints) are recreated on the target table using the same constraint names. However, for unique and primary constraints the underlying index name may be different than the index name on the source table.

Table flags

The table flags of the source table are created on the target table as soon as the target table is created in the INIT phase. These flags are: 'append_mode', 'locksize', 'volatile', 'compression', 'datacapture', 'pctfree', 'logindexbuild', 'owner', 'clustered', and 'droprule'. These flags are then checked at the end of the COPY phase and during the SWAP phase. If there are any changes in the flags they will be updated in the target table.

Grant/Revoke

During the SWAP phase, the stored procedure will go through the entries in SYSCAT.TABAUTH and reproduce the granting of privileges on the table to users/groups/roles.

If the caller of the stored procedure does not have ACCESSCTRL or SECADM authority then the CONTROL privilege cannot be granted. A list of all users/groups/roles that were not granted the CONTROL privilege can be found in the protocol table where the key is WARNINGS.

Please note that if auto_revalidation is enabled on the database, and the USE_AUTO_REVAL option is set (which is the default if auto_revalidation is enabled), then the views will not be dropped as outlined above. Instead, the views will remain and be re-validated with auto_revalidation. Triggers will be dropped and recreated by the stored procedure as there is currently a limitation with renaming a table with a trigger defined as the subject.

Online table move with clustering over an index

It is possible to cluster the target table by an index. If a cluster index is present on the source table, it will be clustered by that index by default. The default can be changed after the INIT phase (This implies phase wise execution of Online Table Move). Calling Online Table Move in one MOVE phase with no cluster index present will result in the stored procedure clustering the target table with the unique/primary index. If a cluster index exists, the stored procedure will cluster the target table using the cluster index.

If there is a cluster index on the source table, it is possible to not cluster the target table on the cluster index by performing a multi-step move and deleting the key entries "COPY_INDEXSCHEMA" and "COPY_INDEXNAME" from the protocol table after the INIT phase.

It is possible to cluster the target table by any secondary index by performing a multi-step move and inserting/updating the key entries "COPY_INDEXSCHEMA" and "COPY_INDEXNAME" in the protocol table with the desired index to cluster the target table.

Changing index attributes

If a user wants to modify the attributes of any existing attributes (i.e. index clustering, index compression, change global to local indexes and vice versa) they can manually make these changes during a multi-step move operation.

This can be done by performing the INIT and COPY phases of the move via a multi-step move. Then manually make any changes to the indexes on the target table. The name of the target table can be found in the protocol table. After the modifications have finished, resume with the REPLAY and SWAP phases..

Restrictions

The following restrictions apply to the ADMIN_MOVE_TABLE stored procedure:

- Only simple tables are supported as the source table. No materialized query tables, typed tables, clustered tables, system tables, views, nicknames, or aliases are permitted.
- A table cannot be moved if an event monitor is currently active on the table.
- Foreign keys (referential constraints) are not supported, either parent or child. To move a table with foreign keys, you can capture the foreign keys using the db2look command, then drop the foreign keys, perform the move operation, and then recreate the keys.

- Tables without a unique index are subject to a complex and potentially expensive replay phase.
- A unique index is required if the table contains LOB, XML, or LONG columns.
- A generated column cannot be part of the MDC specification.
- There is no support for text search indexes.
- Be aware of the large disk space requirements, as the procedure creates two copies of the table and indexes, plus a staging table and log space.
- Copy performance may be an issue as most of the data is moved to the new table using “insert from select” form.
- The VERIFY operation for tables without a unique index does not work on tables with LOBs.
- In releases lower than DB2 Version 9.7 Fix Pack 2, the *DB2_SKIPDELETED* registry variable cannot be set to ON.
- The SYSTOOLSPACE table space must be created and accessible to 'PUBLIC'.
- Lock timeouts are possible during the COPY phase because of long running transactions on the source table.
- Deadlocks can occur during the SWAP phase.
- Deadlocks can occur on a source table with non-unique indexes and several update processes.
- With VARCHAR2 support enabled, the database treats the empty string and NULL as equivalent values, but the single blank is a distinct value. With VARCHAR2 support enabled, the *mdc_cols*, *partkey_cols*, and *data_part* parameters can use a single blank as distinct from the empty string and NULL.
- A table cannot be moved if it is in the Set Integrity Pending state.

Information returned

Table 251. Information returned by the ADMIN_MOVE_TABLE stored procedure

Column name	Data type	Description
TABSCHEMA	VARCHAR(128)	Schema of the table to be moved. Empty string for system wide defaults.
TABNAME	VARCHAR(128)	Table name of the table to be moved. Empty string for system wide defaults.
KEY	VARCHAR(32)	Name of the attribute.
VALUE	CLOB(10M)	Value of the attribute.

The key and value pairs that are returned in the result set can be found in Table 252. To modify the user configurable keys in the result set, use the ADMIN_MOVE_TABLE_UTIL stored procedure.

Table 252. Key and value pairs returned by the ADMIN_MOVE_TABLE stored procedure

Key	Return Value	User Configurable
VERSION	Displays the version of the stored procedure.	No
AUTHID	Displays the authorization ID of the user who called the stored procedure.	No
LOCK	Displays the LOCK start time if another online table move stored procedure call is active, otherwise it is empty.	No

Table 252. Key and value pairs returned by the ADMIN_MOVE_TABLE stored procedure (continued)

Key	Return Value	User Configurable
STATUS	Displays the current status of the online table move: • INIT: INIT is in progress • COPY: COPY is in progress or is possible • REPLAY: REPLAY is in progress or REPLAY and SWAP are possible • CLEANUP: MOVE is complete, but cleanup has not finished or CLEANUP is possible • COMPLETE: MOVE and CLEANUP are complete • COMPLETE_WITH_WARNINGS: MOVE and CLEANUP are complete, however there are warnings (listed under the WARNINGS key).	No
STAGING	Displays the name of the staging table.	No
TARGET	Displays the name of the target table.	No
ORIGINAL	Displays the name of original table after the swap.	No
INDEXSCHEMA	Displays the schema of the index or the empty string if the table does not have an index.	No
INDEXNAME	Displays the name of the index or the empty string if the table does not have an index.	No
COMMIT_AFTER_N_ROWS	During the COPY phase, a commit is executed after this many rows are copied. 0 means no commits during COPY. Default value is 10000.	Yes
DEEPCOMPRESSION_SAMPLE	If the source table has compression enabled, this field specifies how much data (in KB) is sampled when creating a dictionary for compression. 0 means no sampling is done. Default value is 20MB (20480 KB).	Yes
COPY_ARRAY_SIZE	Specifies the ARRAY size for COPY_ARRAY_INSERT. A value less than or equal to 0 means do not use COPY_ARRAY_INSERT. Default value is 100.	Yes
COPY_OPTS	The copy options used during the COPY phase.	No
COPY_INDEXSCHEMA	The schema of the index used to cluster the data on the target table during the COPY phase. This value must be set before the COPY phase. The default schema is the schema name of a cluster index on the source table, if it exists; otherwise the schema name of the unique or primary index on the source table.	Yes
COPY_INDEXNAME	The name of the index used to cluster the data on the target table during the COPY phase. This value must be set before the COPY phase. The default name is the name of a cluster index on the source table, if it exists; otherwise the name of the unique or primary index on the source table.	Yes
INDEX_CREATION_TOTAL_TIME	Displays the total time required for creating secondary indexes.	No
INIT_START	Displays the INIT phase start time.	No
INIT_END	Displays the INIT phase end time.	No
COPY_START	Displays the COPY phase start time.	No
COPY_END	Displays the COPY phase end time.	No

Table 252. Key and value pairs returned by the ADMIN_MOVE_TABLE stored procedure (continued)

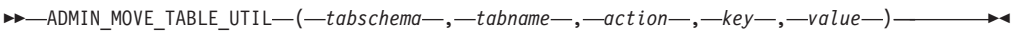
Key	Return Value	User Configurable
COPY_TOTAL_ROWS	Displays the total number of rows copied during the COPY phase.	No
REPLAY_START	Displays the REPLAY phase start time.	No
REPLAY_END	Displays the REPLAY phase end time.	No
REPLAY_TOTAL_ROWS	Displays the accumulated number of replayed rows.	No
REPLAY_TOTAL_TIME	Displays the accumulated time in seconds used for replaying rows.	No
REPLAY_MAX_ERR_RETRIES	Specifies the maximum retry count for errors (lock timeouts or deadlocks) that may occur during the REPLAY phase. Default value is 100.	Yes
REPLAY_THRESHOLD	For a single iteration of the REPLAY phase, if the number of rows applied to the staging table is less than this value, then REPLAY stops, even if new entries are made in the meantime. Default value is 100.	Yes
REORG_USE_TEMPSPACE	If you call the REORG option, you can also specify a temporary table space for the USE clause of the REORG command. If a value is not specified here, the REORG command uses the same table space as the table being reorganized.	Yes
VERIFY_START	Displays the verification start time.	No
VERIFY_END	Displays the verification end time.	No
SWAP_START	Displays the SWAP phase start time.	No
SWAP_END	Displays the SWAP phase end time.	No
SWAP_MAX_RETRIES	Specifies the maximum number of retries allowed during the SWAP phase (if lock timeouts or deadlocks occur). Default value is 10.	Yes
SWAP_RETRIES	Displays the number of retries performed during SWAP phase.	No
CLEANUP_START	Displays the CLEANUP phase start time.	No
CLEANUP_END	Displays the CLEANUP phase end time.	No
WARNINGS	Displays warnings to pass on to the user. These warnings include: • Revalidation of all failed objects • Control could not be granted to a user, group, or role • An index was not created because a column it references no longer exists	No

ADMIN_MOVE_TABLE_UTIL procedure - Modify the online move table procedure

The ADMIN_MOVE_TABLE_UTIL procedure works in conjunction with the SYSPROC.ADMIN_MOVE_TABLE stored procedure when moving active table data. This stored procedure provides a mechanism to alter the user definable values in the ADMIN_MOVE_TABLE protocol table, which is created and used by the ADMIN_MOVE_TABLE procedure.

This procedure will only modify a value in the ADMIN_MOVE_TABLE protocol table if a table move for the table referenced by the TABSCHEMA and TABNAME parameters is already in progress, and the authorization ID of the caller of the procedure is the same as the user executing the table move.

Syntax



The schema for this stored procedure is SYSPROC.

Procedure parameters

tabschema
This input parameter specifies the name of the schema containing the table being moved. This name is case sensitive. and has a data type of VARCHAR(128).

tabname
This input parameter specifies the name of the table being moved. This parameter is case sensitive and has a data type of VARCHAR(128)

action
This input parameter specifies the action for the procedure to execute.

Valid values are:

- UPSERT: If the specified TABSCHEMA.TABNAME.KEY exists in the ADMIN_MOVE_TABLE protocol table, this updates the corresponding VALUE with the new *value* parameter. Otherwise, this inserts the KEY and VALUE pair into the ADMIN_MOVE_TABLE protocol table.
- DELETE: If the specified TABSCHEMA.TABNAME. KEY exists in the ADMIN_MOVE_TABLE protocol table, this deletes the specified KEY and VALUE pair from the ADMIN_MOVE_TABLE protocol table.

This parameter has a datatype of VARCHAR(128).

key
This input parameter specifies the key that to "upsert" or delete in the ADMIN_MOVE_TABLE protocol table.

Valid values are:

- COMMIT_AFTER_N_ROWS: During the COPY phase, a commit is executed after this many rows are copied. A value of 0 means no commits are executed during COPY.
- DEEPCOMPRESSION_SAMPLE: If the source table has compression enabled, this field specifies how much data (in KB) is sampled when creating a dictionary for compression. A value of 0 means no sampling is done.
- COPY_ARRAY_SIZE: Specifies the ARRAY size for COPY_ARRAY_INSERT, a value less than or equal to 0 means do not use COPY_ARRAY_INSERT.
- COPY_INDEXSCHEMA: The schema of the index used to cluster the data on the target table during the COPY phase.
- COPY_INDEXNAME: The name of the index used to cluster the data on the target table during the COPY phase.
- REPLAY_MAX_ERR_RETRIES: Specifies the maximum retry count for errors (lock timeouts or deadlocks) that may occur during the REPLAY phase.

- REPLAY_THRESHOLD: For a single iteration of the REPLAY phase, if the number of rows applied to the staging table is less than this value, then REPLAY stops, even if new entries are made in the meantime.
- REORG_USE_TEMPSPACE: If you call the REORG option in the table move, you can also specify a temporary table space for the USE clause of the REORG command. If a value is not specified here, the REORG command uses the same table space as the table being reorganized.
- SWAP_MAX_RETRIES: Specifies the maximum number of retries allowed during the SWAP phase (if lock timeouts or deadlocks occur).

This parameter has a data type of VARCHAR(128).

value
This input parameter specifies the value to "upsert" into the ADMIN_MOVE_TABLE protocol table. This parameter has a data type of CLOB(10M). The parameter can be NULL or the empty string.

Authorization

No explicit authorization is required to invoke the stored procedure; however, the authorization ID used must be the same as the one used to call the ADMIN_MOVE_TABLE stored procedure.

Examples

This example covers a basic call to the stored procedure in order to update the compression value and remove the specific index information used for the target table copying.

First, the ADMIN_MOVE_TABLE procedure is called to start the table move process before calling the ADMIN_MOVE_TABLE_UTIL procedure in order to update or delete a value in the ADMIN_MOVE_TABLE protocol table:

```
CALL SYSPROC.ADMIN_MOVE_TABLE('SVALENTI','T1','','','','','','','','','INIT')
```

Next, update the DEEP_COMPRESSION_SAMPLE value to 30720 KB:

```
CALL SYSPROC.ADMIN_MOVE_TABLE_UTIL('SVALENTI','T1','UPSERT',
     'DEEPCOMPRESSION_SAMPLE','30720')
```

Now, delete the COPY_INDEXSCHEMA and COPY_INDEXNAME values:

```
CALL SYSPROC.ADMIN_MOVE_TABLE_UTIL('SVALENTI','T1','DELETE','COPY_INDEXSCHEMA','')
CALL SYSPROC.ADMIN_MOVE_TABLE_UTIL('SVALENTI','T1','DELETE','COPY_INDEXNAME','')
```

After these changes, continue the ADMIN_MOVE_TABLE procedure using the new values in the meta table:

```
CALL SYSPROC.ADMIN_MOVE_TABLE('SVALENTI','T1','','','','','','','','','COPY')
CALL SYSPROC.ADMIN_MOVE_TABLE('SVALENTI','T1','','','','','','','','','REPLAY')
CALL SYSPROC.ADMIN_MOVE_TABLE('SVALENTI','T1','','','','','','','','','SWAP')
```

Usage notes

More information regarding the changeable KEY values in the ADMIN_MOVE_TABLE protocol table is available in the Usage notes section of the ADMIN_MOVE_TABLE procedure.

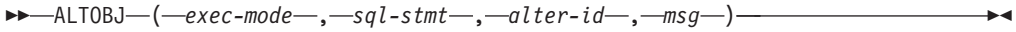
ALTOBJ

The ALTOBJ procedure parses an input CREATE TABLE statement that serves as the target data definition language (DDL) for an existing table that is to be altered. The procedure backs up the data of the table being altered, then drops the original table and creates a new version using the DDL statement; the final step loads the stored data back into the new table.

This procedure supports the following alter table operations and maintains recoverable dependencies:

- Renaming a column
- Increasing or decreasing the size of a column
- Altering a column type and transforming existing data using DB2 scalar functions
- Changing the precision or the scale of decimal values
- Changing the default value of a column
- Changing the nullability attribute of a column to nullable
- Dropping a column

Syntax



The schema is SYSPROC.

Procedure parameters

exec-mode
An input argument of type VARCHAR(30) that specifies one of the following execution modes:

'GENERATE'
Specifies that all the scripts required by the VALIDATE, APPLY, and UNDO modes are to be generated.

'VALIDATE'
Specifies that the statement syntax is to be validated. This option also generates a script to manage the processing of related objects and relationships for the table that is to be altered.

'APPLY_CONTINUE_ON_ERROR' or 'APPLY_STOP_ON_ERROR'
Specifies that a script to manage the processing of related objects and relationships for the table that is to be altered is to be generated. Data from the original table is to be exported, transformed, and used to populate the new table.

'UNDO'
Specifies that any changes made by the alter table operation are to be undone, in case a rollback operation cannot recover errors that might have occurred. This mode is only possible if the original table and any generated scripts have not been deleted.

'FINISH'
Specifies that the renamed original table is to be dropped.

sql-stmt
An input argument of type VARCHAR(2048) that specifies a CREATE TABLE statement that will be used as a template for altering an existing table. When *exec-mode* is 'GENERATE', *sql-stmt* must not be the null value. Otherwise, *sql-stmt* can be the null value, but only if *alter-id* is not -1.

alter-id
An input and output argument of type INTEGER that identifies all of the statements that are generated by this call. If -1 is specified, a new identifier will be generated and returned to the caller. Any existing statements identified by the specified integer are overwritten.

msg
An output argument of type VARCHAR(2048) containing an SQL query that you can execute to display all of the SQL statements generated for or used by the alter table process under the specified execution mode.

Authorization

EXECUTE privilege on the ALTOBJ procedure.

DBADM with LOAD authority, and SETSESSIONUSER are also required.

Examples

Example 1: Run the ALTOBJ procedure to alter column CL2 in table T1 from type INTEGER to BIGINT. The original data definition language for table T1 is:

```
CREATE TABLE T1 (CL1 VARCHAR(5), CL2 INTEGER)
```

The ALTOBJ procedure call to alter the column data type is:

```
CALL SYSPROC.ALTOBJ('APPLY_CONTINUE_ON_ERROR',
   'CREATE TABLE T1 (CL1 VARCHAR(5), CL2 BIGINT)', -1, ?)
```

Note: If you see the following error, try to increase the APPLHEAPSZ parameter value:
SQL0443N Routine "SYSPROC.ALTOBJ" (specific name "ALTOBJ") has returned an error SQLSTATE with diagnostic text "SQL0954 ". SQLSTATE=38553

Example 2: Run the ALTOBJ procedure in VALIDATE mode with *alter-id* input.

```
CALL SYSPROC.ALTOBJ('VALIDATE', CAST (NULL AS VARCHAR(2048)), 123, ?)
```

Usage notes

Although the procedure drops and recreates the table, the user who created the original table will remain as the table definer. However, an audit will show that the table has been dropped and recreated by the user running the procedure.

This procedure does not support the following alter table operations:

- Altering materialized query tables (MQTs) is not supported. Altering a table which contains an MQT is supported.
- Altering typed tables is not supported.
- Altering a remote table using a nickname is not supported.
- Column sequence cannot be reordered.
- Adding and removing, or renaming and removing columns in one call to the procedure is not supported, but adding and renaming columns is supported.

This is because the only way to indicate how the table is to be altered is by the use of the target DDL, rather than column matching information. The following rules are followed by the ALTOBJ procedure when transforming data from the existing table to the altered table:

1. If the number of columns in the existing table is the same as the altered table, it is assumed that no columns are being added or removed. The columns in this case can only be renamed, and are matched by column index.
2. If the number of columns in the existing table is less than in the altered table, it is assumed that columns are being added. The columns can be renamed, and the new columns are added at the end. The existing columns are matched by index.
3. If the number of columns in the existing table is greater than in the altered table, it is assumed that columns are being removed. The columns cannot be renamed and matched by name. The column that is being dropped can be any existing column in the table.

- Structured type UDTs and Reference type UDTs are not supported.
- MQTs defined on a base table which is altered are not populated during the alter table process.

If a table is altered using the ALTOBJ procedure, and the table has an MQT defined, the MQT will be created, but it will not be populated with data.

If a table is altered using the ALTOBJ procedure, and the table has an MQT defined, any columns that are not part of the select result from the table being altered are lost because the MQT content is rebuilt from the new base table.

The definition of the objects might change between ALTOBJ procedure calls because there are no object locks that persist through different sessions.

The table profiles (such as runstats profile) that are associated with the table are lost after going through this extensive alter process.

The SYSTOOLSPACE is used for the routine's operation tables to store metadata; that is, data used to describe database objects and their operation.

APPLICATION_ID

The APPLICATION_ID function returns the application ID of the current connection. The data type of the result is VARCHAR(128).

The value returned by the function is unique within a 100-year interval and valid only for the duration of the connection established before calling the function.

Syntax

The schema is SYSFUN.

Example

```
SELECT APPLICATION_ID() AS APPL_ID FROM SYSIBM.SYSDUMMY1
```

COMPILATION_ENV table function - Retrieve compilation environment elements

The COMPILATION_ENV table function returns the elements of a compilation environment.

Syntax

The schema is SYSPROC.

Table function parameter

compilation-env
An input argument of type BLOB(2M) that contains a compilation environment obtained from the **comp_env_desc** (compilation environment) monitor element.

The function returns a table of two columns (see Table 253 on page 982): NAME VARCHAR(256) and VALUE VARCHAR(1024). The possible values for the compilation environment element names are described in Table 254 on page 982.

The origin of the element values depends primarily on whether the SQL statement is issued dynamically or bound as part of a package.

The number and types of entries in a compilation environment can change over time as capabilities are added to the DB2 database manager. If the compilation environment is from a different DB2 database manager level than the level on which this function is executing, only those elements that are recognized by the level of the function are returned. The descriptions of the elements might also vary from release to release.

Examples

Example 1: Request all the elements of a specific compilation environment that was previously captured by a deadlock event monitor. A deadlock event monitor that is created specifying the WITH DETAILS HISTORY option will capture the compilation environment for dynamic SQL statements. This captured environment is what is accepted as input to the table function.

```
SELECT NAME, VALUE
  FROM TABLE(SYSPROC.COMPILATION_ENV(:hv1)) AS t
```

Example 2: Request a specific element (the default schema) of a compilation environment.

```
SELECT NAME, VALUE
  FROM TABLE(SYSPROC.COMPILATION_ENV(:hv1)) AS t
  WHERE NAME = 'SCHEMA'
```

Example 3: Display the compilation environment for a specific statement in the package cache.

1. Obtain the executable ID, which is used to identify the statement of interest, using the following statement:

```
SELECT EXECUTABLE_ID, VARCHAR{STMT_TEXT, 100)
  FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL,NULL,NULL,-1)) AS t
```

The following is an example output after executing the preceding statement:

```
EXECUTABLE_ID                                                                                                                    2
-------------------------------------------------------------------------------- --------------------------------------------------
x'0100000000000000001000000000000000000000002002009091415140524170 0' select count(*) from syscat.tables
...
```

2. Investigate the compilation environment for the statement (identified using the executable ID) and format the compilation environment using the COMPILATION_ENV table function. The following statement is an example of how this can be done:

```
SELECT VARCHAR(NAME, 30), VARCHAR(VALUE, 50)
  FROM TABLE(COMPILATION_ENV((SELECT COMP_ENV_DESC FROM TABLE
    (MON_GET_PKG_CACHE_STMT(NULL,
     x'0100000000000000001000000000000000000000002002009091415140524170 0',
     NULL, -1)) AS t))) AS s
```

The following is an example output after executing the preceding statement:

```
1                            2
---------------------------- --------------------------------------------------
ISOLATION                    CS
QUERY_OPTIMIZATION           5
MIN_DEC_DIV_3                NO
DEGREE                       1
SQLRULES                     DB2
REFRESH_AGE                  +00000000000000.000000
RESOLUTION_TIMESTAMP         2009-09-14-15.14.05.000000
FEDERATED_ASYNCHRONY         0
PATH                         "SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","SWALKTY"
MAINTAINED_TABLE_TYPE        SYSTEM

  10 record(s) selected.
```

Information returned

Table 253. Information returned by the COMPILATION_ENV table function

Column name	Data type	Description
NAME	VARCHAR(256)	Element of compilation environment. See Table 254 for more details.
VALUE	VARCHAR(1024)	Value of the element.

Table 254. Elements of a compilation environment returned by the COMPILATION_ENV table function

Element name	Description
ISOLATION	The isolation level passed to the SQL compiler. The value is obtained from either the CURRENT ISOLATION special register or the ISOLATION bind option of the current package.
QUERY_OPTIMIZATION	The query optimization level passed to the SQL compiler. The value is obtained from either the CURRENT QUERY OPTIMIZATION special register or the QUERYOPT bind option of the current package.
MIN_DEC_DIV_3	The requested decimal computational scale passed to the SQL compiler. The value is obtained from the *min_dec_div_3* database configuration parameter.
DEGREE	The requested degree of intra-parallelism passed to the SQL compiler. The value is obtained from either the CURRENT DEGREE special register or the DEGREE bind option of the current package.

Table 254. Elements of a compilation environment returned by the COMPILATION_ENV table function (continued)

Element name	Description
SQLRULES	The requested SQL statement behaviors passed to the SQL compiler. The value is derived from the setting of the LANGLVL bind option of the current package. The possible values are 'DB2' or 'SQL92'.
REFRESH_AGE	The allowable data latency passed to the SQL compiler. The value is obtained from either the CURRENT REFRESH AGE special register or the REFRESHAGE bind option of the current package.
SCHEMA	The default schema passed to the SQL compiler. The value is obtained from either the CURRENT SCHEMA special register or the QUALIFIER bind option of the current package.
PATH	The function path passed to the SQL compiler. The value is obtained from either the CURRENT PATH special register or the FUNC_PATH bind option of the current package.
TRANSFORM_GROUP	The transform group information passed to the SQL compiler. The value is obtained from either the CURRENT DEFAULT TRANSFORM GROUP special register or the TRANSFORMGROUP package bind option.
MAINTAINED_TABLE _TYPE	An indicator of what table types can be considered for optimization, passed to the SQL compiler. The value is obtained from the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.
RESOLUTION _TIMESTAMP	The timestamp that is to be used by the SQL compiler for resolving items such as function and data type references in an SQL statement. This timestamp is either the current timestamp or the timestamp of the last explicit bind operation for the current package.
FEDERATED _ASYNCHRONY	The requested degree of federated asynchrony parallelism passed to the SQL compiler. The value is obtained from either the CURRENT FEDERATED ASYNCHRONY special register or the FEDERATED_ASYNCHRONY bind option of the current package.

CONTACTGROUPS administrative view - Retrieve the list of contact groups

The CONTACTGROUPS administrative view returns the list of contact groups, which can be defined locally on the system or in a global list. The setting of the Database Administration Server (DAS) CONTACT_HOST configuration parameter determines whether the list is local or global.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the CONTACTGROUPS administrative view
- CONTROL privilege on the CONTACTGROUPS administrative view

- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the ADMIN_GET_CONTACTGROUPS table function
- DATAACCESS authority

Example

Retrieve all contact group lists.

```
SELECT * FROM SYSIBMADM.CONTACTGROUPS
```

The following is an example of output for this query.

```
NAME          DESCRIPTION                    MEMBERNAME        MEMBERTYPE
------...--- ------------------------...--- ----------...--- ----------
group1        DBA Group1 Contact List        name1             CONTACT
group1        DBA Group1 Contact List        name9             CONTACT
group2        DBA Group2 List                name2             CONTACT
group3                                       group2            GROUP
group5        DBA Group5                     group2            GROUP
group6        DBA Group6                     group3            GROUP
group7                                       name1             CONTACT

  7 record(s) selected.
```

Usage note

The DAS must have been created and be running.

Information returned

Table 255. Information returned by the CONTACTGROUPS administrative view

Column name	Data type	Description
NAME	VARCHAR(128)	Name of the contact group.
DESCRIPTION	VARCHAR(128)	Description of the contact group.
MEMBERNAME	VARCHAR(128)	Name of the member in the contact group. This name can refer to a contact or another contact group.
MEMBERTYPE	VARCHAR(7)	Type of member in the contact group. The type is either CONTACT or GROUP.

CONTACTS administrative view - Retrieve list of contacts

The CONTACTS administrative view returns the list of contacts defined on the database server. The setting of the Database Administration Server (DAS) CONTACT_HOST configuration parameter determines whether the list is local or global.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the CONTACTS administrative view
- CONTROL privilege on the CONTACTS administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the ADMIN_GET_CONTACTS table function
- DATAACCESS authority

Example

Retrieve all contacts.

```
SELECT * FROM SYSIBMADM.CONTACTS
```

The following is an example of output for this query.

```
NAME      TYPE  ADDRESS              MAX_PAGE_LENGTH DESCRIPTION
-----...- ----- ----------------...- --------------- ------------------...-
user1     EMAIL user3@ca.ibm.com                   - DBA Extraordinaire
user2     EMAIL user2@ca.ibm.com                   - DBA on Email
user3     PAGE  user3@ca.ibm.com                 128 DBA on Page
user5     EMAIL user2@ca.ibm.com                   - DBA Extraordinaire

  4 record(s) selected.
```

Usage note

The DAS must have been created and be running.

Information returned

Table 256. Information returned by the CONTACTS administrative view

Column name	Data type	Description
NAME	VARCHAR(128)	Name of contact.
TYPE	VARCHAR(5)	Type of contact: • 'EMAIL' • 'PAGE'
ADDRESS	VARCHAR(128)	SMTP mailbox address of the recipient. For example, joe@somewhere.org.
MAX_PAGE_LENGTH	INTEGER	Maximum message length. Used for example, if the paging service has a message-length restriction.
DESCRIPTION	VARCHAR(128)	Description of contact.

DB_HISTORY administrative view - Retrieve history file information

The DB_HISTORY administrative view returns information from the history files from all database partitions.

You can use the PRUNE HISTORY command on database partitions to reduce the amount of information returned by the DB_HISTORY view. You can also use the LIST HISTORY command to retrieve history information for select database partitions.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the DB_HISTORY administrative view
- CONTROL privilege on the DB_HISTORY administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the ADMIN_LIST_HIST table function
- DATAACCESS authority

Usage note

When a data partitioned table is reorganized, one record for each reorganized data partition is returned. If only a specific data partition of a data partitioned table is reorganized, only a record the for the partition is returned.

Example

Select the database partition number, entry ID, operation, start time, and status information from the database history files for all the database partitions of the database to which the client is currently connected.

```
SELECT DBPARTITIONNUM, EID, OPERATION, START_TIME, ENTRY_STATUS
   FROM SYSIBMADM.DB_HISTORY
```

The following is an example of output for this query.

```
DBPARTITIONNUM EID                  OPERATION START_TIME     ENTRY_STATUS
-------------- -------------------- --------- -------------- ------------
             0                    1 A         20051109185510 A

  1 record(s) selected.
```

Information returned

Table 257. Information returned by the DB_HISTORY administrative view

Column name	Data type	Description
DBPARTITIONNUM	SMALLINT	Database partition number.
EID	BIGINT	Number that uniquely identifies an entry in the history file.
START_TIME	VARCHAR(14)	Timestamp marking the start of a logged event.
SEQNUM	SMALLINT	Sequence number.
END_TIME	VARCHAR(14)	Timestamp marking the end of a logged event.

Table 257. Information returned by the DB_HISTORY administrative view (continued)

Column name	Data type	Description
FIRSTLOG	VARCHAR(254)	Name of the earliest transaction log associated with an event.
LASTLOG	VARCHAR(254)	Name of the latest transaction log associated with an event.
BACKUP_ID	VARCHAR(24)	Backup identifier or unique table identifier.
TABSCHEMA	VARCHAR(128)	Table schema.
TABNAME	VARCHAR(128)	Table name.
COMMENT	VARCHAR(254)	System-generated comment text associated with a logged event.
CMD_TEXT	CLOB(2 M)	Data definition language associated with a logged event.
NUM_TBSPS	INTEGER	Number of table spaces associated with a logged event.
TBSPNAMES	CLOB(5 M)	Names of the table spaces associated with a logged event.
OPERATION	CHAR(1)	Operation identifier. See Table 258 on page 989 for possible values.
OPERATIONTYPE	CHAR(1)	Action identifier for an operation. See Table 258 on page 989 for possible values.
OBJECTTYPE	CHAR(1)	Identifier for the target object of an operation. The possible values are: D for full database, P for table space, and T for table.
LOCATION	VARCHAR(255)	Full path name for files, such as backup images or load input file, that are associated with logged events.

Table 257. Information returned by the DB_HISTORY administrative view (continued)

Column name	Data type	Description
DEVICETYPE	CHAR(1)	Identifier for the device type associated with a logged event. This field determines how the LOCATION field is interpreted. The possible values are: A for TSM, C for client, D for disk, F for snapshot backup, K for diskette, L for local, N (generated internally by DB2), O for other (for other vendor device support), P for pipe, Q for cursor, R for remote fetch data, S for server, T for tape, U for user exit, and X for X/Open XBSA interface.
ENTRY_STATUS	CHAR(1)	Identifier for the status of an entry in the history file. The possible values are: A for active, D for deleted (future use), E for expired, I for inactive, N for not yet committed, Y for committed or active.
SQLCAID	VARCHAR(8)	An "eye catcher" for storage dumps containing 'SQLCA', as it appears in the SQLCAID field of the SQL communications area (SQLCA).
SQLCABC	INTEGER	Length of the SQLCA, as it appears in the SQLCABC field of the SQLCA.
SQLCODE	INTEGER	SQL return code, as it appears in the SQLCODE field of the SQLCA.
SQLERRML	SMALLINT	Length indicator for SQLERRMC, as it appears in the SQLERRML field of the SQLCA.
SQLERRMC	VARCHAR(70)	Contains one or more tokens, separated by X'FF', as they appear in the SQLERRMC field of the SQLCA. These tokens are substituted for variables in the descriptions of error conditions.

Table 257. Information returned by the DB_HISTORY administrative view (continued)

Column name	Data type	Description
SQLERRP	VARCHAR(8)	A three-letter identifier indicating the product, followed by five alphanumeric characters indicating the version, release, and modification level of the product, as they appear in the SQLERRP field of the SQLCA.
SQLERRD1	INTEGER	See "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*.
SQLERRD2	INTEGER	See "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*.
SQLERRD3	INTEGER	See "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*.
SQLERRD4	INTEGER	See "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*.
SQLERRD5	INTEGER	See "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*.
SQLERRD6	INTEGER	See "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*.
SQLWARN	VARCHAR(11)	A set of warning indicators, each containing a blank or 'W'. See "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*.
SQLSTATE	VARCHAR(5)	A return code that indicates the outcome of the most recently executed SQL statement, as it appears in the SQLSTATE field of the SQLCA.

Table 258. OPERATION and OPERATIONTYPE values

Operation value	Operation value description	Operation type
A	Add table space	None
B	Backup	Operation types are: • D = delta offline • E = delta online • F = offline • I = incremental offline • N = online • O = incremental online
C	Load copy	None

Table 258. OPERATION and OPERATIONTYPE values (continued)

Operation value	Operation value description	Operation type
D	Dropped table	None
F	Rollforward	Operation types are: • E = end of logs • P = point in time
G	Reorganize table	Operation types are: • F = offline • N = online
L	Load	Operation types are: • I = insert • R = replace
N	Rename table space	None
O	Drop table space	None
Q	Quiesce	Operation types are: • S = quiesce share • U = quiesce update • X = quiesce exclusive • Z = quiesce reset
R	Restore	Operation types are: • F = offline • I = incremental offline • N = online • O = incremental online • R = rebuild
T	Alter table space	Operation types are: • C = add containers • R = rebalance
U	Unload	None
X	Archive logs	Operation types are: • F = fail archive path • M = mirror log path • N = forced truncation via ARCHIVE LOG command • P = primary log path • 1 = first log archive method • 2 = second log archive method

DBPATHS administrative view - Retrieve database paths

The DBPATHS administrative view returns the values for database paths required for tasks such as split mirror backups.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:
- SELECT privilege on the DBPATHS administrative view
- CONTROL privilege on the DBPATHS administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:
- EXECUTE privilege on the ADMIN_LIST_DB_PATHS table function
- DATAACCESS authority

Example

Retrieve all database paths.

```
SELECT * FROM SYSIBMADM.DBPATHS
```

The following is an example of output for this query.

```
DBPARTITIONNUM TYPE                   ...
-------------- ------------------------...
             0 LOGPATH                ...
             0 MIRRORLOGPATH          ...
             0 DB_STORAGE_PATH        ...
             0 DB_STORAGE_PATH        ...
             0 TBSP_CONTAINER         ...
             0 TBSP_CONTAINER         ...
             0 TBSP_CONTAINER         ...
             0 TBSP_DIRECTORY         ...
             0 TBSP_DIRECTORY         ...
             0 LOCAL_DB_DIRECTORY     ...
             0 DBPATH                 ...

  11 record(s) selected.
```

Output for this query (continued).

```
... PATH
... ------------------------------------------------
... S:\dbfiles\INST5\NODE0000\SQL00001\SQLOGDIR\
... S:\mirrorlogs\NODE0000\
... S:\dbfiles\
... S:\dbfile2\
... S:\dbfiles\INST5\NODE0000\SQL00001\TS3
... S:\dbfiles\INST5\NODE0000\SQL00001\long3
... S:\dbfiles\INST5\NODE0000\SQL00001\regular05
... S:\dbfiles\INST5\NODE0000\SQL00001\usertemp3\
... S:\dbfiles\INST5\NODE0000\SQL00001\systemp3\
... S:\dbfiles\INST5\NODE0000\SQLDBDIR\
... S:\dbfiles\INST5\NODE0000\SQL00001\
```

ADMIN_LIST_DB_PATHS table function

The ADMIN_LIST_DB_PATHS table function returns the list of files required for backup mechanisms such as split mirror backup.

Syntax

The schema is SYSPROC.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the ADMIN_LIST_DB_PATHS table function
- DATAACCESS authority

In addition, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

The ADMIN_LIST_DB_PATHS table function can be invoked as follows:

```
SELECT DBPARTITIONNUM, TYPE, PATH FROM TABLE(ADMIN_LIST_DB_PATHS()) AS FILES
```

The following is an example of output from this query.

```
DBPARTITIONNUM    TYPE            PATH
------------------------------------------------------------------------------------------------
0                 TBSP_CONTAINER  C:\tablespaces\dms\dms1
0                 TBSP_CONTAINER  C:\tablespaces\dms\dms2
1                 TBSP_CONTAINER  C:\tablespaces\dms\dms3
1                 TBSP_DIRECTORY  D:\tablespaces\sms\sms1\
2                 TBSP_DIRECTORY  D:\tablespaces\sms\sms2\
2                 TBSP_DIRECTORY  D:\tablespaces\sms\sms3\
0                 LOGPATH         C:\DB2\NODE0000\SQL00004\SQLOGDIR\
0                 DBPATH          C:\DB2\NODE0000\SQL00004\
1                 LOGPATH         C:\DB2\NODE0001\SQL00004\SQLOGDIR\
1                 DBPATH          C:\DB2\NODE0001\SQL00004\
2                 LOGPATH         C:\DB2\NODE0002\SQL00004\SQLOGDIR\
2                 DBPATH          C:\DB2\NODE0002\SQL00004\
```

If the storage library performing the split mirror operation treats files and directories on RAW devices differently than on normal file systems, you can use the following two queries to obtain, first, the list for all locations on RAW devices:

```
SELECT DBPARTITIONNUM, TYPE, PATH FROM TABLE(ADMIN_LIST_DB_PATHS()) AS FILES
   WHERE TYPE LIKE '%_DEVICE%'
```

Second, the list of files and directories on regular file systems:

```
SELECT DBPARTITIONNUM, TYPE, PATH FROM TABLE(ADMIN_LIST_DB_PATHS()) AS FILES
    WHERE TYPE NOT LIKE '%_DEVICE%'
```

Information returned

Table 259. Information returned by the DBPATHS administrative view and the ADMIN_LIST_DB_PATHS table function

Column name	Data type	Description
DBPARTITIONNUM	SMALLINT	Database partition number.

Table 259. Information returned by the DBPATHS administrative view and the ADMIN_LIST_DB_PATHS table function (continued)

Column name	Data type	Description
TYPE	VARCHAR(64)	Describes the type of database object that the path belongs to. For example the path to the log directory indicated by the LOGPATH database configuration parameter would be shown in this column as LOGPATH. See Table 260 for a list of possible return values.
PATH	VARCHAR(5000)	Path to location where the database manager has a file or directory located. If the path ends with the file system delimiter ('/' on UNIX environments, '\' on Windows environments), the path points to a directory.

Table 260. TYPE column values

Type value	Description
TBSP_DEVICE	Raw device for a database managed space (DMS) table space.
TBSP_CONTAINER	File container for a DMS table space.
TBSP_DIRECTORY	Directory for a system managed space (SMS) table space.
LOGPATH	Primary log path.
LOGPATH_DEVICE	Raw device for primary log path.
MIRRORLOGPATH	Database configuration mirror log path.
DB_STORAGE_PATH	Automatic storage path.
DBPATH	Database directory path.
LOCAL_DB_DIRECTORY	Path to the local database directory.

- For table spaces using automatic storage, both used and unused storage paths are returned. The unused automatic storage paths are needed in case the split mirror backup is restored. Consider the following example: A split mirror backup is taken on a production system. After the backup completes, the automatic storage paths that were not in use before the backup are now in use in production. Assume that there is now a need to restore the split mirror backup. At this point, it is necessary to roll forward the logs from the production database. In order to roll forward the logs, all of the automatic storage paths are required, because all automatic storage paths are now in use.
- Table space containers managed by automatic storage are not returned individually. Instead, they are reflected in the automatic storage path column.
- The automatic storage paths are returned once per database partition.
- The values returned for LOGPATH and MIRRORLOGPATH are the values stored in memory. Changed values stored on disk, which are only applicable after a database restart, are not returned.

- If output from `SELECT * FROM SYSIBMADM.DBPATHS` is being used to create a db2relocatedb configuration file (a file containing the configuration information necessary for relocating a database), the DBPATH output must be modified appropriately before it can be used in the configuration file.

  For example, the following DBPATH output:

  ```
  /storage/svtdbm3/svtdbm3/NODE0000/SQL00001/
  ```

  can be used to specify the DB_PATH parameter in a db2relocatedb configuration file as follows:

  ```
  DB_PATH=/storage/svtdbm3,/storage_copy2/svtdbm3
  ```
- The LOCAL_DB_DIRECTORY path might contain information belonging to multiple databases. Because the sqldbdir is not separated for multiple databases created in the same directory, ensure that the target system to which files will be copied does not have any databases already existing in that path.
- If two or more databases share at least one automatic storage path, the split mirror operation for one of these databases might affect more than one database, causing I/O problems for the databases that were not intended to be split.

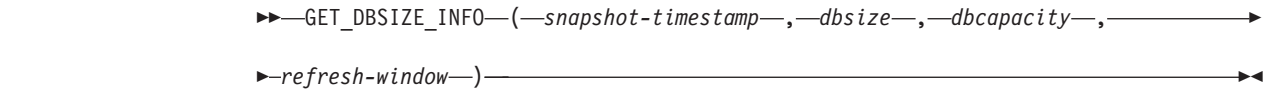
Restriction

This administrative view cannot be called when the database is in WRITE SUSPEND mode. The database administrator must ensure that the physical layout of the database does not change in the time between the invocation of the view and the activation of the WRITE SUSPEND mode, which is needed to perform the split mirror operation. The split mirror backup image might not be restored successfully if, for example, the table space layout changed in that time.

GET_DBSIZE_INFO

The GET_DBSIZE_INFO procedure calculates the database size and maximum capacity.

Syntax



The schema is SYSPROC.

Procedure parameters

snapshot-timestamp
An output parameter of type TIMESTAMP that returns the time at which *dbsize* and *dbcapacity* were calculated. This timestamp, along with the value of *refresh-window*, is used to determine when the cached values in the SYSTOOLS.STMG_DBSIZE_INFO table need to be refreshed.

dbsize
An output parameter of type BIGINT that returns the size of the database (in bytes). The database size is calculated as follows: *dbsize* = sum (used_pages * page_size) for each table space (SMS & DMS).

dbcapacity
An output parameter of type BIGINT that returns the database capacity (in

bytes). This value is not available on partitioned database systems. The database capacity is calculated as follows: *dbcapacity* = SUM (DMS usable_pages * page size) + SUM (SMS container size + file system free size per container). If multiple SMS containers are defined on the same file system, the file system free size is included only once in the calculation of capacity.

refresh-window
An input argument of type INTEGER that specifies the number of minutes until the cached values for database size and capacity are to be refreshed. Specify -1 for the default refresh window of 30 minutes. A refresh window of 0 forces an immediate refreshing of the cached values.

Authorization

- SYSMON authority
- EXECUTE privilege on the GET_DBSIZE_INFO procedure

Examples

Example 1: Get the database size and capacity using a default refresh window of 30 minutes. The database size and capacity will be recalculated when the cached data is older than 30 minutes.

```
CALL GET_DBSIZE_INFO(?, ?, ?, -1)
```

The procedure returns:

```
 Value of output parameters
 --------------------------
 Parameter Name  : SNAPSHOTTIMESTAMP
 Parameter Value : 2004-02-29-18.31.55.178000

 Parameter Name  : DATABASESIZE
 Parameter Value : 22302720

 Parameter Name  : DATABASECAPACITY
 Parameter Value : 4684793856

 Return Status = 0
```

Example 2: Get the database size and capacity using a refresh window of 0 minutes. The database size and capacity will be recalculated immediately.

```
CALL GET_DBSIZE_INFO(?, ?, ?, 0)
```

The procedure returns:

```
Value of output parameters
--------------------------
Parameter Name  : SNAPSHOTTIMESTAMP
Parameter Value : 2004-02-29-18.33.34.561000

Parameter Name  : DATABASESIZE
Parameter Value : 22302720

Parameter Name  : DATABASECAPACITY
Parameter Value : 4684859392

 Return Status = 0
```

Example 3: Get the database size and capacity using a refresh window of 24 hours. The database size and capacity will be recalculated when the cached data is older than 1440 minutes.

```
CALL GET_DBSIZE_INFO(?, ?, ?, 1440)
```

The procedure returns:

```
Value of output parameters
--------------------------
Parameter Name  : SNAPSHOTTIMESTAMP
Parameter Value : 2004-02-29-18.33.34.561000

Parameter Name  : DATABASESIZE
Parameter Value : 22302720

Parameter Name  : DATABASECAPACITY
Parameter Value : 4684859392

Return Status = 0
```

Usage notes

The calculated values are returned as procedure output parameters and are cached in the SYSTOOLS.STMG_DBSIZE_INFO table. The procedure caches these values because the calculations are costly. The SYSTOOLS.STMG_DBSIZE_INFO table is created automatically the first time the procedure executes. If there are values cached in the SYSTOOLS.STMG_DBSIZE_INFO table and they are current enough, as determined by the *snapshot-timestamp* and *refresh-window* values, these cached values are returned. If the cached values are not current enough, new cached values are calculated, inserted into the SYSTOOLS.STMG_DBSIZE_INFO table and returned, and the *snapshot-timestamp* value is updated.

To ensure that the data is returned by all partitions for a global table space snapshot, the database must be activated.

The SYSTOOLSPACE is used for the routine's operation tables to store metadata; that is, data used to describe database objects and their operation.

NOTIFICATIONLIST administrative view - Retrieve contact list for health notification

The NOTIFICATIONLIST administrative view returns the list of contacts and contact groups that are notified about the health of an instance.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the NOTIFICATIONLIST administrative view
- CONTROL privilege on the NOTIFICATIONLIST administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the HEALTH_GET_NOTIFICATION_LIST table function
- DATAACCESS authority

Example

Retrieve all contacts that will receive notification of health alerts.

```
SELECT * FROM SYSIBMADM.NOTIFICATIONLIST
```

The following is an example of output for this query.

```
NAME                 TYPE
-------------...------- -------
group3               GROUP
user4                CONTACT
group3               GROUP

  3 record(s) selected.
```

Information returned

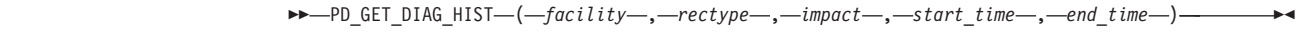
Table 261. Information returned by the NOTIFICATIONLIST administrative view

Column name	Data type	Description
NAME	VARCHAR(128)	Name of contact.
TYPE	VARCHAR(7)	Type of contact: • 'CONTACT' • 'GROUP'

PD_GET_DIAG_HIST - Return records from a given facility

The PD_GET_DIAG_HIST table function returns log records, event records and notification records from a given facility. Options are also supported to filter based on the type of record, customer impact value of the record and from-until timestamps.

Syntax



The schema is SYSPROC.

Table function parameters

facility
An optional input argument of type VARCHAR(20) that specifies the facility from which records are to be returned. A facility is a logical grouping that records relate to. The possible values are:

- ALL: Returns records from all facilities
- MAIN: Returns records from the DB2 general diagnostic logs. This currently means the db2diag log files, the admin notification log, and the rotating event logs.
- OPTSTATS: Return records related to optimizer statistics

If this parameter is null or an empty string (''), 'ALL' is the default.

rectype
An optional input argument of type VARCHAR(30) that specifies which record type to return. A combination of types separated by '+' are supported, for example: 'D + EI'. The possible values are:

- 'ALL': Return all record types.
- 'D': Return all diagnostic records.
- 'E': Return all event records.

- 'DI': Internal diagnostic records. These are non-translated diagnostic record that are used by IBM support in a diagnostic situation.
- 'DX': External diagnostic records. These are translated diagnostic that are of use to the user. These records are the notification records.
- 'EI': Internal event record. These are event record that are used by IBM support in a diagnostic situation.
- 'EX': External event record. These are diagnostic record that are of use to the user.

If this parameter is null or an empty string (''), all records are returned.

impact
An optional input argument of type VARCHAR(18) that specifies the minimum customer impact level of the record returned. The possible values are:

- 'NONE'
- 'UNLIKELY'
- 'POTENTIAL'
- 'IMMEDIATE'
- 'CRITICAL'

If this parameter is null or an empty string (''), all records are returned.

start_time
An optional input argument of type TIMESTAMP that specifies a valid timestamp. Entries are returned if their timestamp is more recent than this value. If this parameter is null or an empty string (''), records are returned regardless of how old they are.

end_time
An optional input argument of type TIMESTAMP that specifies a valid timestamp. Entries are returned if their timestamp is older than this value. If this parameter is null or an empty string (''), records are returned regardless of how recent they are.

Authorization

EXECUTE privilege on the PD_GET_DIAG_HIST table function.

Example

```
SELECT FACILITY, RECTYPE, TIMESTAMP, IMPACT, SUBSTR(MSG,1, 50) AS MSG
FROM TABLE (PD_GET_DIAG_HIST( 'MAIN', 'E', '', CAST (NULL AS TIMESTAMP),
  CAST (NULL AS TIMESTAMP) ) ) AS T
WHERE T.PROCESS_NAME = 'db2star2' OR T.PROCESS_NAME = 'db2stop2'
```

The following is an example of output from this query.

```
FACILITY            RECTYPE TIMESTAMP                  ...
------------------- ------- -------------------------- ...
MAIN                EX      2007-06-25-11.34.05.756171 ...
MAIN                EX      2007-06-25-11.34.25.946646 ...

  2 record(s) selected.
```

Output from this query (continued).

```
... IMPACT             MSG
... ------------------ --------------------------------------------------
... -                  ADM7514W  Database manager has stopped.
... -                  ADM7513W  Database manager has started.
```

Information returned

Table 262. Information returned by the PD_GET_DIAG_HIST table function

Column Name	Data Type	Description
FACILITY	VARCHAR(20)	A facility is a logical grouping which records relate to. The possible values are: • ALL: Returns records from all facilities • MAIN: Returns records from the DB2 general diagnostic logs. This currently means the db2diag log files, the admin notification log, and the rotating event logs. • OPTSTATS: Return records related to optimizer statistics
RECTYPE	VARCHAR(3)	The type of record. The possible values are: • 'DI': Internal diagnostic record • 'DX': External diagnostic record • 'EI': Internal event record • 'EX': External event record
TIMESTAMP	TIMESTAMP	The time that the message was created.
TIMEZONE	INTEGER	The time difference (in minutes) from the Universal Coordinated Time (UCT). For example, -300 is EST.
INSTANCENAME	VARCHAR(128)	The name of the instance where the message was created.
DBPARTITIONNUM	SMALLINT	The partition number where the message was created. For non-partitioned database, 0 is returned.
LEVEL	CHAR(1)	The severity level of the record. The possible values are: • 'C': Critical • 'E': Error • ' I': Informational • 'S': Severe • 'W': Warning
IMPACT	VARCHAR(18)	Qualifies the impact of this message from a user's perspective. This clarifies the impact of the message on the business process DB2 is part of. The possible values are: • 'CRITICAL' • 'IMMEDIATE' • 'NONE' • 'POTENTIAL' • 'UNLIKELY'
DBNAME	VARCHAR(128)	The name of the database being accessed while this message was created.
EDU_ID	BIGINT	The Engine Dispatched Unit identifier that created this message.
EDUNAME	VARCHAR(64)	The name of the engine Dispatched Unit that created this message.
PID	BIGINT	The operating system process identifier that created this message.
PROCESS_NAME	VARCHAR(255)	The operating system process name that created this message.

Table 262. Information returned by the PD_GET_DIAG_HIST table function (continued)

Column Name	Data Type	Description
TID	BIGINT	The thread numerical identifier that created this message.
APPLNAME	VARCHAR(255)	The name of the client application that initiated the connection, if it is available.
APPL_ID	VARCHAR(64)	The application identifier that initiated the connection if available. For example: 'G91A3955.F33A.02DD18143340'
APPLHANDLE	VARCHAR(9)	A system-wide unique identifier for the application that initiated the connection when available. This is synonymous to agent ID. The identifier consists of the coordinating partition number and a 16-bit counter separated by a '-'. The format is as follows: 'nnn-xxxxx'
AUTH_ID	VARCHAR(30)	The system authorization identifier of the process.
PRODUCT	VARCHAR(50)	The name of the product that created the message. For example 'DB2 Common'.
COMPONENT	VARCHAR(255)	The name of the component that created the message.
FUNCTION	VARCHAR(255)	The name of the function that generated the message.
PROBE	INTEGER	Probe point number used to identify where the message was generated in the function.
CALLEDPRODUCT	VARCHAR(50)	The name of the product at the source of the error. This is used when the source of an error is not where the message was created.
CALLEDCOMPONENT	VARCHAR(255)	The name of the component at the source of the error. This is used when the source of an error is not where the message was created.
CALLEDFUNCTION	VARCHAR(255)	The name of the function at the source of the error. This is used when the source of an error is not where the message was created.
OSERR	INTEGER	The operating system error number.
RETCODE	INTEGER	The product specific return code.
MSGNUM	INTEGER	The numeric message number for the associated message, if it is available. For example, this is the numerical portion of ADM7513W.
MSGTYPE	CHAR(3)	The type related to the message identifier, if it is available. For example, ADM is used for administration notification log messages.
MSG	CLOB(16KB)	The short description text for this record. This is the translated message text corresponding to the MSGNUM, and MSGTYPE for translated messages. For non-translated messages, this is the short description. For example : 'Bringing down all db2fmp processes as part of db2stop'.

Table 262. Information returned by the PD_GET_DIAG_HIST table function (continued)

Column Name	Data Type	Description
OBJTYPE	VARCHAR(64)	The type of object the event applies to, if it is available. The possible values are: • 'APM' • 'CATALOG CACHE ENTRY' • 'CFG' • 'CLI' • 'CLP' • 'CONTAINER' • 'COUNTER' • 'DAS' • 'DB2AGENT' • 'DB PART MAP ID' • 'DB PART NUM' • 'DBA' • 'DBM' • 'DMS' • 'DPS' • 'EDU' • 'EVALUATION' • 'EXTENDER' • 'FCM' • 'HISTOGRAM TEMPLATE' • 'INDEX STATS' • 'INITIAL SAMPLING' • 'REDIST DB PART GROUP' • 'REDIST TABLE' • 'RDS' • 'SAMPLING TEST' • 'SERVICE CLASS' • 'STATS' • 'STATS DAEMON' • 'TABLE' • 'TABLE STATS' • 'TABLE AND INDEX STATS' • 'THRESHOLD' • 'UDF' • 'WORK ACTION SET' • 'WORK CLASS SET' • 'WORKLOAD'
OBJNAME	VARCHAR(255)	The name of the object the event relates to, if it is available.
OBJNAME_QUALIFIER	VARCHAR(255)	Additional information about the object, if it is available.

Table 262. Information returned by the PD_GET_DIAG_HIST table function (continued)

Column Name	Data Type	Description
EVENTTYPE	VARCHAR(24)	The event type is the action or verb associated with this event. The possible values are: • 'ACCEPT' • 'ACCESS' • 'ADD' • 'ALTER' • 'ASSOCIATE' • 'AVAILABLE' • 'BRINGDOWN' • 'CHANGE' • 'CHANGECFG' • 'CLOSE' • 'COLLECT' • 'CONNECT' • 'CREATE' • 'DEPENDENCY' • 'DESTROY' • 'DISASSOCIATE' • 'DISCONNECT' • 'DISPATCH' • 'DROP' • 'FINI' • 'FREE' • 'GET' • 'INIT' • 'INTERRUPT' • 'OPEN','READ' • 'RECV' • 'REPLY' • 'REPORT' • 'REQUEST' • 'RESET' • 'SEND' • 'START' • 'STARTUP' • 'STOP' • 'SWITCH' • 'TERMINATE' • 'TRANSFER' • 'WAIT' • 'WORK' • 'WRITE'
EVENTDESC	VARCHAR(256)	A short representation of the key fields for this event.

Table 262. Information returned by the PD_GET_DIAG_HIST table function (continued)

Column Name	Data Type	Description
FIRST_EVENTQUALIFIERTYPE	VARCHAR(64)	The type of the first event qualifier. Event qualifiers are used to describe what was affected by the event. The possible values are: • 'AT' • 'BY' • 'CONTEXT' • 'DUE TO' • 'FOR' • 'FROM' • 'ON' • 'TO' If *facility* is OPTSTATS, the only value is 'AT'.
FIRST_EVENTQUALIFIER	CLOB(16K)	The first qualifier for the event. If *facility* is OPTSTATS, this will be a timestamp indicating when the statistics collection occurred.
SECOND_EVENTQUALIFIERTYPE	VARCHAR(64)	The type of the second event qualifier. If *facility* is OPTSTATS, the value is 'BY'.
SECOND_EVENTQUALIFIER	CLOB(16K)	The second qualifier for the event. If *facility* is OPTSTATS, the possible values are: • Asynchronous • FABRICATE • FABRICATE PARTIAL • SYNCHRONOUS • SYNCHRONOUS SAMPLED • USER
THIRD_EVENTQUALIFIERTYPE	VARCHAR(64)	The type of the third event qualifier. If *facility* is OPTSTATS, the value is 'DUE TO'.
THIRD_EVENTQUALIFIER	CLOB(16K)	The third qualifier for the event. If *facility* is OPTSTATS, the possible values are: • Conflict • Error • Object unavailable • RUNSTATS error • Timeout
EVENTSTATE	VARCHAR(255)	State of the object or action as a result of the event. This can also contain a percentage indicating the progression of the event.

Table 262. Information returned by the PD_GET_DIAG_HIST table function (continued)

Column Name	Data Type	Description
EVENTATTRIBUTE	VARCHAR(255)	The event attributes. This is a list of attributes associated with the event. when more than one attribute is used, the list is separated by '+' characters. For example 'CACHED + LOGICAL + AUTO'. The possible values are: • 'ASYNC' • 'AUTO' • 'CACHED' • 'DIRECT' • 'EXTERNAL' • 'INDIRECT' • 'INTERNAL' • 'LOGICAL' • 'PERMANENT' • 'PHYSICAL' • 'SYNC' • 'TEMPORARY'
EVENTSTACK	CLOB(16K)	The logical event stack at the point the record was logged when applicable.
CALLSTACK	CLOB(16K)	The operating system stack dump for the thread that generated this record when applicable.
DUMPFILE	CLOB(5000)	The name of the secondary dump file associated with the log record when applicable. This is a fully qualified path to a file or directory where additional information related to the message can be retrieved.
FULLREC	CLOB(16K)	Formatted text version of the entire record. This section also contains additional DATA fields.

PDLOGMSGS_LAST24HOURS administrative view and PD_GET_LOG_MSGS table function – Retrieve problem determination messages

The PDLOGMSGS_LAST24HOURS administrative view and the PD_GET_LOG_MSGS table function return problem determination log messages that were logged in the DB2 notification log. The information is intended for use by database and system administrators.

PDLOGMSGS_LAST24HOURS administrative view

The PDLOGMSGS_LAST24HOURS administrative view returns problem determination log messages that were logged in the DB2 notification log in the last 24 hours.

The schema is SYSIBMADM.

Refer to Table 263 on page 1010 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the PDLOGMSGS_LAST24HOURS administrative view
- CONTROL privilege on the PDLOGMSGS_LAST24HOURS administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the PD_GET_LOG_MSGS table function
- DATAACCESS authority

Example

Get all critical log messages logged in the last 24 hours, ordered by most recent.

```
SELECT * FROM SYSIBMADM.PDLOGMSGS_LAST24HOURS
   WHERE MSGSEVERITY = 'C' ORDER BY TIMESTAMP DESC
```

The following is an example of output from this query.

```
TIMESTAMP                  TIMEZONE    INSTANCENAME      ...
-------------------------- ----------- ----------------- ...
2005-11-23-21.56.41.240066        -300 svtdbm4           ...
                                                         ...
                                                         ...
                                                         ...
                                                         ...
                                                         ...
                                                         ...
2005-11-23-21.56.39.150597        -300 svtdbm4           ...
2005-11-23-21.56.37.363384        -300 svtdbm4           ...
                                                         ...
                                                         ...
                                                         ...
2005-11-23-21.56.35.880314        -300 svtdbm4           ...
                                                         ...


  4 record(s) selected.
```

Output from this query (continued).

```
...  DBPARTITIONNUM DBNAME        PID         PROCESSNAME          ...
...  -------------- ------------- ---...------ -------------------- ...
...               0 CAPTAIN           4239374 db2agent (CAPTAIN) 0 ...
...                                                                ...
...                                                                ...
...                                                                ...
...                                                                ...
...                                                                ...
...                                                                ...
...               0 CAPTAIN           4239374 db2agent (CAPTAIN) 0 ...
...               0 CAPTAIN           4239374 db2agent (CAPTAIN) 0 ...
...                                                                ...
...                                                                ...
...                                                                ...
...               0 CAPTAIN           4239374 db2agent (CAPTAIN) 0 ...
...                                                                ...
...                                                                ...
```

Output from this query (continued).

```
...TID APPL_ID                          COMPONENT            ...
...--- ------------------------------ -------------------- ...
...  1 9.26.15.148.36942.051124025612  oper system services  ...
...                                                         ...
...                                                         ...
...                                                         ...
...                                                         ...
...                                                         ...
...                                                         ...
...  1 9.26.15.148.36942.051124025612  base sys utilities    ...
...  1 9.26.15.148.36942.051124025612  relation data serv    ...
...                                                         ...
...                                                         ...
...                                                         ...
...  1 9.26.15.148.36942.051124025612  relation data serv    ...
...                                                         ...
...                                                         ...
```

Output from this query (continued).

```
... FUNCTION           PROBE  MSGNUM      MSGTYPE ...
... ------------------ ------ ----------- ------- ...
... sqloSleepInstance      38         504 ADM     ...
...                                               ...
...                                               ...
...                                               ...
...                                               ...
...                                               ...
...                                               ...
... sqleMarkDBad           10        7518 ADM     ...
... sqlrr_dump_ffdc        10           1 ADM     ...
...                                               ...
...                                               ...
...                                               ...
... sqlrr_dump_ffdc        10           1 ADM     ...
...                                               ...
```

Output from this query (continued).

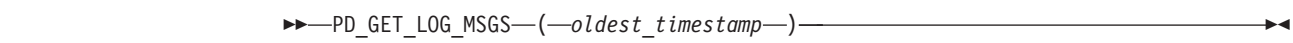
```
...  MSGSEVERITY MSG
...  ----------- -----------------------------------
...  C           ADM0504C  An unexpected internal
...              processing error has occurred.  ALL
...              DB2 PROCESSES ASSOCIATED WITH THIS
...              INSTANCE HAVE BEEN SUSPENDED.
...              Diagnostic information has been
...              recorded.  Contact IBM Support
...              for further assistance.
...  C           ADM7518C  "CAPTAIN " marked bad.
...  C           ADM0001C  A severe error has occurred.
...              Examine the administration notification
...              log and contact IBM Support if
...              necessary.
...  C           ADM0001C  A severe error has occurred.
...              Examine the administration notification
...              log and contact IBM Support if necessary.
```

PD_GET_LOG_MSGS table function

The PD_GET_LOG_MSGS table function returns the same information as the PDLOGMSGS_LAST24HOURS administrative view, but allows you to specify a specific time period that is not limited to the last 24 hours.

Refer to Table 263 on page 1010 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

oldest_timestamp
An input argument of type TIMESTAMP that specifies a valid timestamp. Entries are returned starting with the most current timestamp and ending with the log entry with the timestamp specified by this input argument. If a null value is specified, all log entries are returned.

Authorization

EXECUTE privilege on the PD_GET_LOG_MSGS table function.

Examples

Example 1: Retrieve all notification messages logged for database SAMPLE on instance DB2 in the last week for all database partitions. Report messages in chronological order.

```
SELECT TIMESTAMP, APPL_ID, DBPARTITIONNUM, MSG
   FROM TABLE ( PD_GET_LOG_MSGS( CURRENT_TIMESTAMP - 7 DAYS)) AS T
   WHERE INSTANCENAME = 'DB2' AND DBNAME = 'SAMPLE'
   ORDER BY TIMESTAMP ASC
```

The following is an example of output from this query.

```
TIMESTAMP                  APPL_ID                     DBPARTITIONNUM ...
-------------------------- --------------------------- -------------- ...
2005-11-13-12.51.37.772000 *LOCAL.DB2.050324175005                 0 ...
                                                                     ...
2005-11-13-12.51.37.772001 *LOCAL.DB2.050324175005                 0 ...
                                                                     ...
2005-11-13-12.51.37.781000 *LOCAL.DB2.050324175005                 0 ...
                                                                     ...
2005-11-13-12.51.37.781001 *LOCAL.DB2.050324175005                 0 ...
                                                                       ...
                                                                       ...
2005-11-17-14.12.39.036001 *LOCAL.DB2.041117191249                 0 ...
2005-11-17-14.12.39.056000 *LOCAL.DB2.041117191249                 0 ...
2005-11-17-14.13.04.450000 *LOCAL.DB2.041117191307                 0 ...
2005-11-17-14.13.04.460000 *LOCAL.DB2.041117191307                 0 ...
2005-11-17-14.18.29.042000 *LOCAL.DB2.041117190824                 0 ...
...
...
...
```

Output from this query (continued).

```
... MSG
... ------------------------------------------------------------...--
... ADM5502W  The escalation of "143" locks on table
...    "SYSIBM  .SYSINDEXAUTH" to lock intent "X" was successful.
... ADM5502W  The escalation of "144" locks on table
...    "SYSIBM  .SYSINDEXES" to lock intent "X" was successful.
... ADM5502W  The escalation of "416" locks on table
...    "SYSIBM  .SYSINDEXCOLUSE" tolock intent "X" was successful.
... ADM5500W  DB2 is performing lock escalation.  The total
...    number of locks currently held is "1129", and the target
```

```
...    number of locks to hold is "564".
... ADM7506W  Database quiesce has been requested.
... ADM7507W  Database quiesce request has completed successfully.
... ADM7510W  Database unquiesce  has been requested.
... ADM7509W  Database unquiesce request has completed successfully.
... ADM4500W  A package cache overflow condition has occurred.  There
...    is no error but this indicates that the package cache has
...    exceeded the configured maximum size.  If this condition persists,
...    you may want to adjust the PCKCACHESZ DB configuration parameter.
```

Example 2: Retrieve all critical errors logged on instance DB2 for database partition 0 in the last day, sorted by most recent.

```
SELECT TIMESTAMP, DBNAME, MSG
   FROM TABLE (PD_GET_LOG_MSGS(CURRENT_TIMESTAMP - 1 DAYS)) AS T
   WHERE MSGSEVERITY = 'C' AND INSTANCENAME = 'DB2' AND
   DBPARTITIONNUM = 0
   ORDER BY TIMESTAMP DESC
```

The following is an example of output from this query.

```
TIMESTAMP                  DBNAME      MSG
-------------------------- ----------- ------------------------
2004-11-04-13.49.17.022000 TESTSBCS    ADM0503C  An unexpected
                                       internal processing error
                                       has occurred.  ALL DB2
                                       PROCESSES ASSOCIATED WITH
                                       THIS INSTANCE HAVE BEEN
                                       SHUTDOWN.  Diagnostic
                                       information has been
                                       recorded.  Contact IBM
                                       Support for further
                                       assistance.
2004-11-04-11.32.26.760000 SAMPLE      ADM0503C  An unexpected
                                       internal processing error
                                       has occurred.  ALL DB2
                                       PROCESSES ASSOCIATED WITH
                                       THIS INSTANCE HAVE BEEN
                                       SHUTDOWN.  Diagnostic
                                       information has been
                                       recorded.  Contact IBM
                                       Support for further
                                       assistance.

  2 record(s) selected.
```

Example 3: Retrieve messages written by DB2 processes servicing application with application ID of *LOCAL.DB2.050927195337, over the last day.

```
SELECT TIMESTAMP, MSG
   FROM TABLE (PD_GET_LOG_MSGS(CURRENT_TIMESTAMP - 1 DAYS)) AS T
   WHERE APPL_ID = '*LOCAL.DB2.050927195337'
```

The following is an example of output from this query.

```
TIMESTAMP                  MSG
-------------------------- ----------------------------------------

2005-06-27-21.17.12.389000 ADM4500W  A package cache overflow
                           condition has occurred.  There is no error
                           but this indicates that the package cache
                           has exceeded the configured maximum
                           size.  If this condition persists, you
                           may want to adjust the PCKCACHESZ DB
                           configuration parameter.
2005-06-27-18.41.22.248000 ADM4500W  A package cache overflow
                           condition has occurred.  There is no error
                           but this indicates that the package cache
```

```
                          has exceeded the configured maximum
                          size.  If this condition persists, you
                          may want to adjust the PCKCACHESZ DB
                          configuration parameter.
2005-06-27-12.51.37.772001 ADM5502W  The escalation of "143" locks
                          on table "SYSIBM  .SYSINDEXAUTH" to
                          lock intent "X" was successful.
2005-06-27-12.51.37.772000 ADM5502W  The escalation of "144" locks
                          on table "SYSIBM  .SYSINDEXES" to lock
                          intent "X" was successful.
2005-06-27-12.51.37.761001 ADM5502W  The escalation of "416" locks
                          on table "SYSIBM  .SYSINDEXCOLUSE" to
                          lock intent "X" was successful.

...
```

Example 4: Find all instances of message ADM0504C in the notification log. Note that the messages considered are not limited by a timestamp. This could be an expensive operation if the notification logfile is very large.

```
SELECT TIMESTAMP, DBPARTITIONNUM, DBNAME, MSG
   FROM TABLE (PD_GET_LOG_MSGS(CAST(NULL AS TIMESTAMP))) AS T
   WHERE MSGNUM = 504 AND MSGTYPE = 'ADM' AND MSGSEVERITY = 'C'
```

The following is an example of output from this query.

```
TIMESTAMP                  DBPARTITIONNUM DBNAME       ...
-------------------------  -------------- -------------...
2005-11-23-21.56.41.240066               0 CAPTAIN      ...
...
...
...
...
...
...
...
...
...
```

Output from this query (continued).

```
... APPL_ID                          MSG
... -------------------------------- -------------------------
... 9.26.15.148.36942.051124025612   ADM0504C  An unexpected
...                                  internal processing error
...                                  has occurred.  ALL DB2
...                                  PROCESSES ASSOCIATED WITH
...                                  THIS INSTANCE HAVE BEEN
...                                  SUSPENDED.  Diagnostic
...                                  information has been
...                                  recorded.  Contact IBM
...                                  Support for further
...                                  assistance.
```

Information returned

Note: In a partitioned database environment, the order in which log messages are returned cannot be guaranteed. If the order of log records is important, the results should be sorted by timestamp.

Table 263. Information returned by the PDLOGMSGS_LAST24HOURS administrative view and the PD_GET_LOG_MSGS table function

Column name	Data type	Description
TIMESTAMP	TIMESTAMP	The time when the entry was logged.
TIMEZONE	INTEGER	Time difference (in minutes) from Universal Coordinated Time (UCT). For example, -300 is EST.
INSTANCENAME	VARCHAR(128)	Name of the instance that generated the message.
DBPARTITIONNUM	SMALLINT	The database partition that generated the message. For a non partitioned database environment, 0 is returned.
DBNAME	VARCHAR(128)	The database on which the error or event occurred.
PID	BIGINT	Process ID of the process that generated the message.
PROCESSNAME	VARCHAR(255)	Name of process that generated the message.
TID	BIGINT	ID of the thread within the process that generated the message.
APPL_ID	VARCHAR(64)	ID of the application for which the process is working.
COMPONENT	VARCHAR(255)	The name of the DB2 component that is providing the message. For messages written by user applications using the db2AdminMsgWrite API, "User Application" is returned.
FUNCTION	VARCHAR(255)	The name of the DB2 function that is providing the message. For messages written by user applications using the db2AdminMsgWrite API, "User Function" is returned.
PROBE	INTEGER	Unique internal identifier that allows DB2 Customer Support and Development to locate the point in the DB2 source code that generated the message.
MSGNUM	INTEGER	The numeric message number for the error or event.

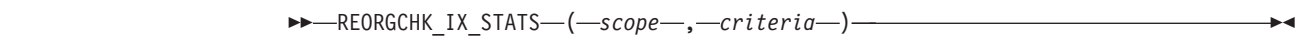
Table 263. Information returned by the PDLOGMSGS_LAST24HOURS administrative view and the PD_GET_LOG_MSGS table function (continued)

Column name	Data type	Description
MSGTYPE	CHAR(3)	Indicates the message type: ADM (for messages written to the administration notification log) or NULL if the message type cannot be determined.
MSGSEVERITY	CHAR(1)	Message severity: C (critical), E (error), W (warning), I (informational) or NULL (if the message severity could not be determined).
MSG	CLOB(16 KB)	Notification log message text.

REORGCHK_IX_STATS procedure – Retrieve index statistics for reorganization evaluation

The REORGCHK_IX_STATS procedure returns a result set containing index statistics that indicate whether or not there is a need for reorganization.

Syntax



The schema is SYSPROC.

Procedure parameters

scope
An input argument of type CHAR(1) that specifies the scope of the tables that are to be evaluated, using one of the following values:

'T' Table

'S' Schema

criteria
An input argument of type VARCHAR(259). If *scope* has a value of 'T', specifies a fully qualified table name, or accepts one of the following values: ALL, USER, or SYSTEM. If *scope* has a value of 'S', specifies a schema name.

Authorization

- SELECT privilege on catalog tables.
- EXECUTE privilege on the REORGCHK_IX_STATS procedure.

Example

```
CALL SYSPROC.REORGCHK_IX_STATS('T','JESCOTT.EMPLOYEE')
```

Usage note

The procedure uses the SYSTOOLSTMPSPACE table space. If SYSTOOLSTMPSPACE does not already exist, the procedure will create this table space.

Information returned

Table 264. Information returned by the REORGCHK_IX_STATS procedure

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema name.
TABLE_NAME	VARCHAR(128)	Table name.
DATAPARTITIONNAME	VARCHAR(128)	Name of the data partition. NULL for nonpartitioned tables.
INDEX_SCHEMA	VARCHAR(128)	Index schema name.
INDEX_NAME	VARCHAR(128)	Index name.
INDCARD	BIGINT	Number of index entries in the index. This can be different than table cardinality for some indexes. For example, the index cardinality on XML columns might be greater than the table cardinality.
NLEAF	BIGINT	Total number of index leaf pages.
NUM_EMPTY_LEAFS	BIGINT	Number of pseudo-empty index leaf pages.
NLEVELS	INTEGER	Number of index levels.
NUMRIDS_DELETED	BIGINT	Number of pseudo-deleted RIDs.
FULLKEYCARD	BIGINT	Number of unique index entries that are not marked deleted.
LEAF_RECSIZE	BIGINT	Record size of the index entry on a leaf page. This is the average size of the index entry excluding any overhead and is calculated from the average column length of all columns participating in the index.
NONLEAF_RECSIZE	BIGINT	Record size of the index entry on a non-leaf page. This is the average size of the index entry excluding any overhead and is calculated from the average column length of all columns participating in the index except any INCLUDE columns.
LEAF_PAGE_OVERHEAD	BIGINT	Reserved space on the index leaf page for internal use.
NONLEAF_PAGE_OVERHEAD	BIGINT	Reserved space on the index non-leaf page for internal use
PCT_PAGES_SAVED	SMALLINT	Percent of pages saved using Index Compression. A non-zero number indicates the index is compressed.
F4	INTEGER	F4 formula value.
F5	INTEGER	F5 formula value.

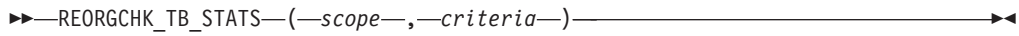
Table 264. Information returned by the REORGCHK_IX_STATS procedure (continued)

Column name	Data type	Description
F6	INTEGER	F6 formula value.
F7	INTEGER	F7 formula value.
F8	INTEGER	F8 formula value.
REORG	CHAR(5)	A 5-character field, each character mapping to one of the five formulas: F4, F5, F6, F7, and F8; a dash means that the formula value is in the recommended range; an asterisk means that the formula value is out of the recommended range, indicating a need for reorganization.

REORGCHK_TB_STATS procedure – Retrieve table statistics for reorganization evaluation

The REORGCHK_TB_STATS procedure returns a result set containing table statistics that indicate whether or not there is a need for reorganization.

Syntax



The schema is SYSPROC.

Procedure parameters

scope
An input argument of type CHAR(1) that specifies the scope of the tables that are to be evaluated, using one of the following values:

'T' Table

'S' Schema

criteria
An input argument of type VARCHAR(259). If *scope* has a value of 'T', specifies a fully qualified table name, or accepts one of the following values: ALL, USER, or SYSTEM. If *scope* has a value of 'S', specifies a schema name.

Authorization

- SELECT privilege on catalog tables.
- EXECUTE privilege on the REORGCHK_TB_STATS procedure.

Example

```
CALL SYSPROC.REORGCHK_TB_STATS('T','JESCOTT.EMPLOYEE')
```

Usage note

The procedure uses the SYSTOOLSTMPSPACE table space. If SYSTOOLSTMPSPACE does not already exist, the procedure will create this table space.

Information returned

Table 265. Information returned by the REORGCHK_TB_STATS procedure

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema name.
TABLE_NAME	VARCHAR(128)	Table name.
DATAPARTITIONNAME	VARCHAR(128)	Name of the data partition. NULL for nonpartitioned tables.
CARD	BIGINT	Cardinality (number of rows in the table).
OVERFLOW	BIGINT	Number of overflow rows.
NPAGES	BIGINT	Total number of pages on which the rows of the table exist; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
FPAGES	BIGINT	Total number of pages; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
ACTIVE_BLOCKS	BIGINT	Total number of active blocks for a multidimensional clustering (MDC) table. This field is only applicable to tables defined using the ORGANIZE BY clause. It indicates the number of blocks of the table that contains data.
TSIZE	BIGINT	Size of the table.
F1	INTEGER	F1 formula value.
F2	INTEGER	F2 formula value.
F3	INTEGER	F3 formula value.
REORG	CHAR(3)	A 3-character field, each character mapping to one of the three formulas: F1, F2, and F3; a dash means that the formula value is in the recommended range; an asterisk means that the formula value is out of the recommended range, indicating a need for reorganization

SQLERRM scalar functions - Retrieves error message information

There are two versions of the SQLERRM scalar function. The first allows for full flexibility of message retrieval including using message tokens and language selection. The second takes only an SQLCODE as an input parameter and returns the short message in English.

SQLERRM scalar function

This SQLERRM scalar function takes a message identifier, locale and token input and returns the short or long message of type VARCHAR(32672) in the specified locale. If the input locale is not supported by the server, the message is returned in English.

Syntax

The schema is SYSPROC.

Scalar function parameters

msgid
An input argument of type VARCHAR(9) that represents the message number for which the information should be retrieved. The message number is the application return code prefixed with 'SQL', 'DBA' or 'CLI'. For example, 'SQL551', 'CLI0001'. The message number can also be an SQLSTATE, for example, '42829'.

tokens
An input argument of type VARCHAR(70) that represents the error message token list. Some messages might not have tokens. If this parameter is null, then no token replacement occurs in the returned message. Token replacement only occurs when returning the default short messages. If the long message option is selected, no token replacement occurs.

token_delimiter
An input argument of type VARCHAR(1) that represents the token delimiter. This delimiter must be unique and not contained in any tokens passed to the scalar function. If no delimiter is supplied, the default delimiter used is the semicolon.

locale
An input argument of type VARCHAR(33) that represents the locale to pass to the server in order to have the error message retrieved in that language. If no locale is specified, or the server does not support the locale, the message is returned in English and a warning is returned.

shortmsg
An input argument of type INTEGER that is used to indicate if the long message should be returned instead of the default short message. To return long messages, this value must be set to 0 or CAST(NULL as INTEGER).

Authorization

EXECUTE privilege on the SQLERRM scalar function.

Examples

Example 1: Retrieve the English short message for SQL0551N with tokens "AYYANG", "UPDATE" and "SYSCAT.TABLES".

```
VALUES (SYSPROC.SQLERRM
   ('SQL551', 'AYYANG;UPDATE;SYSCAT.TABLES', ';', 'en_US', 1))
```

The following is an example of output returned.

```
1
------------------------------------------------------------...--
SQL0551N "AYYANG" does not have the privilege to perform operation
   "UPDATE" on object "SYSCAT.TABLES"
```

Example 2: Retrieve the English error message associated with SQLSTATE 42501.

```
VALUES (SYSPROC.SQLERRM ('42501', '', '', 'en_US', 1))
```

The following is an example of output returned.

```
1
------------------------------------------------------------...--
SQLSTATE 42501: The authorization ID does not have the privilege to
   perform the specified operation on the identified object.
```

Example 3: Retrieve the English long error message for SQL1001N.

```
VALUES (SYSPROC.SQLERRM ('SQL1001', '', '', 'en_US', 0))
```

The following is an example of output returned.

```
1
------------------------------------------------------------...--
SQL1001N "<name>" is not a valid database name.

Explanation:

The syntax of the database name specified in the command is not
valid.  The database name must contain 1 to 8 characters and all
the characters must be from the database manager base character
set.

 The command cannot be processed.

User Response:

Resubmit the command with the correct database name.

 sqlcode :  -1001

 sqlstate :  2E000
```

SQLERRM scalar function

This SQLERRM scalar function takes an SQLCODE as the only input and returns the short message of type VARCHAR(32672) for the specified SQLCODE in English.

Syntax

The schema is SYSPROC.

Scalar function parameter

sqlcode
An input argument of type INTEGER that represents an SQLCODE.

Authorization

EXECUTE privilege on the SQLERRM scalar function.

Example

Retrieve the short message for SQLCODE SQL0551N.

```
VALUES (SYSPROC.SQLERRM (551))
```

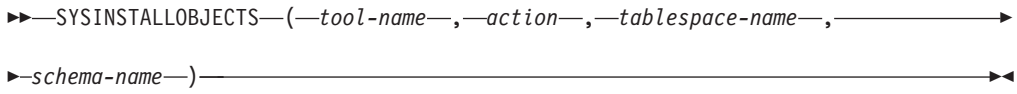
The following is an example of output returned.

```
1
------------------------------------------------------------...--
SQL0551N  "" does not have the privilege to perform operation
   "" on object "".
```

SYSINSTALLOBJECTS

The SYSINSTALLOBJECTS procedure creates or drops the database objects that are required for a specific tool.

Syntax



The schema is SYSPROC.

Procedure parameters

tool-name
An input argument of type VARCHAR(128) that specifies the name of the tool that is to be loaded, using one of the following values:

- 'AM' for creating activity monitor objects
- 'DB2AC' for autonomous computing (health monitor)
- 'STMG_DBSIZE_INFO' for storage management
- 'OPT_PROFILES' for creating the optimization profile table
- 'POLICY' for policy (tables and triggers)
- 'EXPLAIN' for creating or migrating explain tables

action
An input argument of type CHAR(1) that specifies the action that is to be taken. Valid values are:

C Create objects.

D Drop objects.

V Verify objects.

M Migrate objects. The M option is only valid when used with the tool name EXPLAIN. This option migrates explain tables that were created in Version 9.5 through Version 9.7 to be compatible with Version 9.7 Fix Pack 1. Explain tables you created in Version 9.7 Fix Pack 1, or later, are not modified.

tablespace-name
An input argument of type VARCHAR(128) that specifies the name of the table space in which the objects are to be created. If a value is not specified, or the value is an empty or blank string, the default user space is used if the tool name is AM. If the tool name is EXPLAIN and the action is M, the input table space name is ignored and the table space is used where the explain tables that are being migrated were created. Otherwise, the SYSTOOLSPACE table space is used. If SYSTOOLSPACE does not already exist, it will be created.

schema-name
Except for 'EXPLAIN' tool-name option, SYSTOOLS is always used as the schema regardless of the schema-name passed as the input parameter.

For 'EXPLAIN' tool-name option, an input schema-name can be passed and the tables are created under the specified schema-name. If no schema-name is passed as the input parameter, SYSTOOLS schema is used.

Example

```
CALL SYSPROC.SYSINSTALLOBJECTS('AM', 'C', CAST (NULL AS VARCHAR(128)),
   CAST (NULL AS VARCHAR(128)))
```

Chapter 22. Deprecated SQL administrative routines and their replacement routines or views

To provide expanded support in DB2 for Linux, UNIX, and Windows Version 9.7 for the existing administrative routines, some of the DB2 Version 9.5 routines have been replaced with new, more comprehensive routines or views.

Applications that use the DB2 for Linux, UNIX, and Windows Version 9.7 table functions should be modified to use the new functions or administrative views. The new table functions have the same base names as the original functions but are suffixed with '_Vxx' for the version of the product in which they were added (for example, _V97). In most cases, the new table functions and administrative views return additional information. The administrative views will always be based on the most current version of the table functions, and therefore allow for more application portability. As the columns may vary from one release to the next (that is, some are added and some are deleted), it is recommended that specific columns be selected from the administrative views, or that the result set be described if a SELECT * statement is used by an application.

Table 266. Deprecated SQL administrative routines or views and their replacement routines or views for DB2 Version 9.7 FP1

DB2 Version 9.7 deprecated function	New DB2 for Linux, UNIX, and Windows Version 9.7 FP1 function or view
"LOCKS_HELD administrative view - Retrieve information on locks held" on page 561	• **"MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415** • **"MON_GET_LOCKS - list all locks in the currently connected database" on page 442** • **"MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381** • **"MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503**
"LOCKWAITS administrative view - Retrieve current lockwaits information" on page 564	• **"MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415** • **"MON_GET_LOCKS - list all locks in the currently connected database" on page 442** • **"MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381** • **"MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503**
"SNAPLOCK administrative view and SNAP_GET_LOCK table function – Retrieve lock logical data group snapshot information" on page 650	• "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415 • "MON_GET_LOCKS - list all locks in the currently connected database" on page 442 • "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381 • "MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503

Table 266. Deprecated SQL administrative routines or views and their replacement routines or views for DB2 Version 9.7 FP1 (continued)

DB2 Version 9.7 deprecated function	New DB2 for Linux, UNIX, and Windows Version 9.7 FP1 function or view
"LOCKS_HELD administrative view - Retrieve information on locks held" on page 561	• **"MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415** • **"MON_GET_LOCKS - list all locks in the currently connected database" on page 442** • **"MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381** • **"MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503**
"LOCKWAITS administrative view - Retrieve current lockwaits information" on page 564	• **"MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415** • **"MON_GET_LOCKS - list all locks in the currently connected database" on page 442** • **"MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381** • **"MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503**
"SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function – Retrieve lockwait logical data group snapshot information" on page 655	• "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415 • "MON_GET_LOCKS - list all locks in the currently connected database" on page 442 • "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381 • "MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503
"SNAPDB administrative view and SNAP_GET_DB_V95 table function - Retrieve snapshot information from the dbase logical group" on page 749	"SNAPDB administrative view and SNAP_GET_DB_V97 table function - Retrieve snapshot information from the dbase logical group" on page 609

Table 267. Deprecated SQL administrative routines and their replacement routines or views for DB2 Version 9.7

DB2 Version 9.5 deprecated function	New DB2 for Linux, UNIX, and Windows Version 9.7 function or view
"ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO" on page 1029	"ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function - returns compressed information" on page 228
"ADMIN_GET_TAB_INFO table function - Retrieve size and state information for tables" on page 1022	"ADMINTABINFO administrative view and ADMIN_GET_TAB_INFO_V97 table function - retrieve table size and state information" on page 236
"SNAPSTORAGE_PATHS administrative view and SNAP_GET_STORAGE_PATHS table function - Retrieve automatic storage path information" on page 807	"SNAPSTORAGE_PATHS administrative view and SNAP_GET_STORAGE_PATHS_V97 table function - Retrieve automatic storage path information" on page 667
"SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V91 table function - Retrieve tablespace_nodeinfo logical data group snapshot information" on page 832	"SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V97 table function - Retrieve tablespace_nodeinfo logical data group snapshot information" on page 692

Table 267. Deprecated SQL administrative routines and their replacement routines or views for DB2 Version 9.7 (continued)

DB2 Version 9.5 deprecated function	New DB2 for Linux, UNIX, and Windows Version 9.7 function or view
"WLM_GET_ACTIVITY_DETAILS - Return detailed information about a specific activity" on page 1192	"MON_GET_ACTIVITY_DETAILS table function - Get complete activity details" on page 404
"WLM_GET_SERVICE_SUBCLASS_STATS - return statistics of service subclasses" on page 1209	"WLM_GET_SERVICE_SUBCLASS_STATS_V97 table function - Return statistics of service subclasses" on page 928
"WLM_GET_WORKLOAD_STATS - return workload statistics" on page 1220	"WLM_GET_WORKLOAD_STATS_V97 table function - Return workload statistics" on page 944
"WLM_GET_WORKLOAD_OCCURRENCE _ACTIVITIES - Return a list of activities" on page 1215	"WLM_GET_WORKLOAD_OCCURRENCE _ACTIVITIES_V97 - Return a list of activities" on page 939
"WLM_GET_SERVICE_CLASS_WORKLOAD _OCCURRENCES - List of workload occurrences" on page 1205	"WLM_GET_SERVICE_CLASS_WORKLOAD _OCCURRENCES_V97 - List workload occurrences" on page 924
"WLM_GET_SERVICE_CLASS_AGENTS - List agents running in a service class" on page 1199	"WLM_GET_SERVICE_CLASS_AGENTS_V97 table function - List agents running in a service class" on page 916

The health monitor has been deprecated in DB2 for Linux, UNIX, and Windows Version 9.7 . The deprecated health monitor interfaces are still supported in Version 9.7. A new suite of GUI tools for managing DB2 for Linux, UNIX, and Windows data and data-centric applications is available and can be used instead of the Control Center tools. For more information, see Database management and application development tools.

Table 268. Deprecated Health Monitor routines

"HEALTH_CONT_HI" on page 1036
"HEALTH_CONT_HI_HIS" on page 1037
"HEALTH_CONT_INFO" on page 1039
"HEALTH_DB_HI" on page 1041
"HEALTH_DB_HI_HIS" on page 1044
"HEALTH_DB_HIC" on page 1048
"HEALTH_DB_HIC_HIS" on page 1050
"HEALTH_DB_INFO" on page 1052
"HEALTH_DBM_HI" on page 1054
"HEALTH_DBM_HI_HIS" on page 1055
"HEALTH_DBM_INFO" on page 1058
"HEALTH_GET_ALERT_ACTION_CFG" on page 1059
"HEALTH_GET_ALERT_CFG" on page 1062
"HEALTH_GET_IND_DEFINITION" on page 1065
"HEALTH_HI_REC" on page 1067
"HEALTH_TBS_HI" on page 1069
"HEALTH_TBS_HI_HIS" on page 1071
"HEALTH_TBS_INFO" on page 1075

In the previous release, DB2 Version 9.5, there were also new functions that replaced DB2 Version 9.1 functions.

Table 269. Deprecated SQL administrative routines and their replacement routines or views for DB2 Version 9.5

DB2 Version 9.1 deprecated function	New DB2 Version 9.5 function or view
"SNAP_GET_APPL table function – Retrieve appl logical data group snapshot information" on page 1077	"SNAPAPPL administrative view and SNAP_GET_APPL_V95 table function - Retrieve appl logical data group snapshot information" on page 588
"SNAP_GET_APPL_INFO table function – Retrieve appl_info logical data group snapshot information" on page 1084	"SNAPAPPL_INFO administrative view and SNAP_GET_APPL_INFO_V95 table function - Retrieve appl_info logical data group snapshot information" on page 580
"SNAP_GET_BP table function – Retrieve bufferpool logical group snapshot information" on page 1090	"SNAPBP administrative view and SNAP_GET_BP_V95 table function - Retrieve bufferpool logical group snapshot information" on page 596
"SNAP_GET_DB_V91 table function - Retrieve snapshot information from the dbase logical group" on page 1105	"SNAPDB administrative view and SNAP_GET_DB_V95 table function - Retrieve snapshot information from the dbase logical group" on page 749
"SNAP_GET_DBM table function – Retrieve the dbm logical grouping snapshot information" on page 1102	"SNAPDBM administrative view and SNAP_GET_DBM_V95 table function - Retrieve the dbm logical grouping snapshot information" on page 624
"SNAP_GET_DYN_SQL_V91 table function - Retrieve dynsql logical group snapshot information" on page 1127	"SNAPDYN_SQL administrative view and SNAP_GET_DYN_SQL_V95 table function - Retrieve dynsql logical group snapshot information" on page 635

ADMIN_GET_TAB_INFO table function - Retrieve size and state information for tables

Note: This table function has been deprecated and replaced by the "ADMINTABINFO administrative view and ADMIN_GET_TAB_INFO_V97 table function - retrieve table size and state information" on page 236.

The ADMIN_GET_TAB_INFO table function provides methods to retrieve table size and state information that is not currently available in the catalog views.

Refer to the ADMIN_GET_TAB_INFO table function metadata table for a complete list of information that can be returned.

Syntax

The schema is SYSPROC.

Table function parameters

tabschema
An input argument of type VARCHAR(128) that specifies a schema name.

tabname
An input argument of type VARCHAR(128) that specifies a table name, a materialized query table name or a hierarchy table name.

Authorization

EXECUTE privilege on the ADMIN_GET_TAB_INFO table function.

Example

Example 1: Retrieve size and state information for the table DBUSER1.EMPLOYEE.

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_TAB_INFO('DBUSER1', 'EMPLOYEE'))
   AS T
```

Example 2: Suppose there exists a non-partitioned table (DBUSER1.EMPLOYEE), with all associated objects (for example, indexes and LOBs) stored in a single table space. Calculate how much physical space the table is using in the table space:

```
SELECT (data_object_p_size + index_object_p_size + long_object_p_size +
   lob_object_p_size + xml_object_p_size) as total_p_size
   FROM TABLE( SYSPROC.ADMIN_GET_TAB_INFO( 'DBUSER1', 'EMPLOYEE' )) AS T
```

Calculate how much space would be required if the table were moved to another table space, where the new table space has the same page size and extent size as the original table space:

```
SELECT (data_object_l_size + index_object_l_size + long_object_l_size +
   lob_object_l_size + xml_object_l_size) as total_l_size
   FROM TABLE( SYSPROC.ADMIN_GET_TAB_INFO( 'DBUSER1', 'EMPLOYEE' )) AS T
```

Usage notes

- If both the *tabschema* and *tabname* are specified, information is returned for that specific table only.
- If the *tabschema* is specified but *tabname* is empty (") or NULL, information is returned for all tables in the given schema.
- If the *tabschema* is empty (") or NULL and *tabname* is specified, an error is returned. To retrieve information for a specific table, the table must be identified by both schema and table name.
- If both *tabschema* and *tabname* are empty (") or NULL, information is returned for all tables.
- If *tabschema* or *tabname* do not exist, or *tabname* does not correspond to a table name (type T), a materialized query table name (type S) or a hierarchy table name (type H), an empty result set is returned.
- When the ADMIN_GET_TAB_INFO table function is retrieving data for a given table, it will acquire a shared lock on the corresponding row of SYSTABLES to ensure consistency of the data that is returned (for example, to ensure that the table is not dropped while information is being retrieved for it). The lock will only be held for as long as it takes to retrieve the size and state information for the table, not for the duration of the table function call.
- Physical size reported for tables in SMS table spaces is the same as logical size.
- When an inplace reorg is active on a table, the physical size for the data object (DATA_OBJECT_P_SIZE) will not be calculated. Only the logical size will be returned. You can tell if an inplace reorg is active on the table by looking at the INPLACE_REORG_STATUS output column.
- The logical size reported for LOB objects created before DB2 UDB Version 8 might be larger than the physical size if the objects have not yet been reorganized.

ADMIN_GET_TAB_INFO table function metadata

Table 270. ADMIN_GET_TAB_INFO table function metadata

Column name	Data type	Description
TABSCHEMA	VARCHAR(128)	Schema name.
TABNAME	VARCHAR(128)	Table name.
TABTYPE	CHAR(1)	Table type: • 'H' = hierarchy table • 'S' = materialized query table • 'T' = table
DBPARTITIONNUM	SMALLINT	Database partition number.
DATA_PARTITION_ID	INTEGER	Data partition number.
AVAILABLE	CHAR(1)	State of the table: • 'N' = the table is unavailable. If the table is unavailable, all other output columns relating to the size and state will be NULL. • 'Y' = the table is available. **Note:** Rollforward through an unrecoverable load will put a table into the unavailable state.
DATA_OBJECT_L_SIZE	BIGINT	Data object logical size. Amount of disk space logically allocated for the table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated for the table (for example, in the case of a logical table truncation). For multi-dimensional clustering (MDC) tables, this size includes the logical size of the block map object. The size returned takes into account full extents that are logically allocated for the table and, for objects created in DMS table spaces, an estimate of the Extent Map Page (EMP) extents. This size represents the logical size of the base table only. Space consumed by LOB data, Long Data, Indexes and XML objects are reported by other columns.

Table 270. ADMIN_GET_TAB_INFO table function metadata (continued)

Column name	Data type	Description
DATA_OBJECT_P_SIZE	BIGINT	Data object physical size. Amount of disk space physically allocated for the table, reported in kilobytes. For MDC tables, this size includes the size of the block map object. The size returned takes into account full extents allocated for the table and includes the EMP extents for objects created in DMS table spaces. This size represents the physical size of the base table only. Space consumed by LOB data, Long Data, Indexes and XML objects are reported by other columns.
INDEX_OBJECT_L_SIZE	BIGINT	Index object logical size. Amount of disk space logically allocated for the indexes defined on the table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold index data for the table (for example, in the case of a logical table truncation). The size returned takes into account full extents that are logically allocated for the indexes and, for indexes created in DMS table spaces, an estimate of the EMP extents. This value is only reported for non-partitioned tables. For partitioned tables, this value will be 0.
INDEX_OBJECT_P_SIZE	BIGINT	Index object physical size. Amount of disk space physically allocated for the indexes defined on the table, reported in kilobytes. The size returned takes into account full extents allocated for the indexes and includes the EMP extents for indexes created in DMS table spaces. This value is only reported for non-partitioned tables. For partitioned tables this value will be 0.

Table 270. ADMIN_GET_TAB_INFO table function metadata (continued)

Column name	Data type	Description
LONG_OBJECT_L_SIZE	BIGINT	Long object logical size. Amount of disk space logically allocated for long field data in a table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold long field data for the table (for example, in the case of a logical table truncation). The size returned takes into account full extents that are logically allocated for long field data and, for long field data created in DMS table spaces, an estimate of the EMP extents.
LONG_OBJECT_P_SIZE	BIGINT	Long object physical size. Amount of disk space physically allocated for long field data in a table, reported in kilobytes. The size returned takes into account full extents allocated for long field data and includes the EMP extents for long field data created in DMS table spaces.
LOB_OBJECT_L_SIZE	BIGINT	LOB object logical size. Amount of disk space logically allocated for LOB data in a table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold LOB data for the table (for example, in the case of a logical table truncation). The size includes space logically allocated for the LOB allocation object. The size returned takes into account full extents that are logically allocated for LOB data and, for LOB data created in DMS table spaces, an estimate of the EMP extents.
LOB_OBJECT_P_SIZE	BIGINT	LOB object physical size. Amount of disk space physically allocated for LOB data in a table, reported in kilobytes. The size includes space allocated for the LOB allocation object. The size returned takes into account full extents allocated for LOB data and includes the EMP extents for LOB data created in DMS table spaces.

Table 270. ADMIN_GET_TAB_INFO table function metadata (continued)

Column name	Data type	Description
XML_OBJECT_L_SIZE	BIGINT	XML object logical size. Amount of disk space logically allocated for XML data in a table, reported in kilobytes. The logical size is the amount of space that the table knows about. It might be less than the amount of space physically allocated to hold XML data for the table (for example, in the case of a logical table truncation). The size returned takes into account full extents that are logically allocated for XML data and, for XML data created in DMS table spaces, an estimate of the EMP extents.
XML_OBJECT_P_SIZE	BIGINT	XML object physical size. Amount of disk space physically allocated for XML data in a table, reported in kilobytes. The size returned takes into account full extents allocated for XML data and includes the EMP extents for XML data created in DMS table spaces.
INDEX_TYPE	SMALLINT	Indicates the type of indexes currently in use for the table. Returns: • 1 if type-1 indexes are being used. • 2 if type-2 indexes are being used.
REORG_PENDING	CHAR(1)	A value of 'Y' indicates that a reorg recommended alter has been applied to the table and a classic (offline) reorg is required. Otherwise 'N' is returned.
INPLACE_REORG_STATUS	VARCHAR(10)	Current status of an inplace table reorganization on the table. The status value can be one of the following: • ABORTED (in a PAUSED state, but unable to RESUME; STOP is required) • EXECUTING • NULL (if no inplace reorg has been performed on the table) • PAUSED
LOAD_STATUS	VARCHAR(12)	Current status of a load operation against the table. The status value can be one of the following: • IN_PROGRESS • NULL (if there is no load in progress for the table and the table is not in load pending state) • PENDING

Table 270. ADMIN_GET_TAB_INFO table function metadata (continued)

Column name	Data type	Description
READ_ACCESS_ONLY	CHAR(1)	'Y' if the table is in Read Access Only state, 'N' otherwise. A value of 'N' should not be interpreted as meaning that the table is fully accessible. If a load is in progress or pending, a value of 'Y' means the table data is available for read access, and a value of 'N' means the table is inaccessible. Similarly, if the table status is set integrity pending (refer to SYSCAT.TABLES STATUS column), then a value of 'N' means the table is inaccessible.
NO_LOAD_RESTART	CHAR(1)	A value of 'Y' indicates the table is in a partially loaded state that will not allow a load restart. A value of 'N' is returned otherwise.
NUM_REORG_REC_ALTERS	SMALLINT	Number of reorg recommend alter operations (for example, alter operations after which a reorganization is required) that have been performed against this table since the last reorganization.
INDEXES_REQUIRE_ REBUILD	CHAR(1)	'Y' if any of the indexes defined on the table require a rebuild, and 'N' otherwise. If no indexes are defined on the table, 'N' will also be returned, as there are no indexes that require a rebuild.
LARGE_RIDS	CHAR(1)	Indicates whether or not the table is using large row IDs (RIDs) (4 byte page number, 2 byte slot number). A value of 'Y' indicates that the table is using large RIDs and 'N' indicates that it is not using large RIDs. A value of 'P' (pending) will be returned if the table supports large RIDs (that is, the table is in a large table space), but at least one of the indexes for the table has not been reorganized or rebuilt yet, so the table is still using 4 byte RIDs (which means that action must be taken to convert the table or indexes).

Table 270. ADMIN_GET_TAB_INFO table function metadata (continued)

Column name	Data type	Description
LARGE_SLOTS	CHAR(1)	Indicates whether or not the table is using large slots (which allows more than 255 rows per page). A value of 'Y' indicates that the table is using large slots and 'N' indicates that it is not using large slots. A value of 'P' (pending) will be returned if the table supports large slots (that is, the table is in a large table space), but there has been no offline table reorganization or table truncation operation performed on the table yet, so it is still using a maximum of 255 rows per page.
DICTIONARY_SIZE	BIGINT	Size of the dictionary, in bytes, used for row compression if a row compression dictionary exists for the table.

ADMINTABCOMPRESSINFO view and ADMIN_GET_TAB_COMPRESS_INFO

Note: This table function has been deprecated and replaced by the "ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function - returns compressed information" on page 228.

The ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO table function return compression information for tables, materialized query tables (MQT) and hierarchy tables.

ADMINTABCOMPRESSINFO administrative view

The ADMINTABCOMPRESSINFO administrative view returns compression information for tables, materialized query tables (MQT) and hierarchy tables only. These table types are reported as T for table, S for materialized query tables and H for hierarchy tables in the SYSCAT.TABLES catalog view. The information is returned at both the data partition level and the database partition level for a table.

The schema is SYSIBMADM.

Refer to the ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO table function metadata table for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the ADMINTABCOMPRESSINFO administrative view
- CONTROL privilege on the ADMINTABCOMPRESSINFO administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the ADMIN_GET_TAB_COMPRESS_INFO table function
- DATAACCESS authority

Examples

Example 1: Retrieve all compression information for all tables

```
SELECT * FROM SYSIBMADM.ADMINTABCOMPRESSINFO
```

The following is an example of output from this query:

```
TABSCHEMA     TABNAME         DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER      DICT_BUILD_TIMESTAMP
---------...- -----------...- -------------- ----------------- ------------- -------------...- --------------------------
SYSIBM        SYSTABLES                    0                 0 N             NOT BUILT    -
SYSIBM        SYSCOLUMNS                   0                 0 N             NOT BUILT    -
...
SIMAP2        STAFF                        0                 0 Y             REORG        2006-08-27-19.07.36.000000
SIMAP2        PARTTAB                      0                 0 Y             REORG        2006-08-27-22.07.17.000000
...

  156 record(s) selected.
```

Output from this query (continued):

```
COMPRESS_DICT_SIZE   EXPAND_DICT_SIZE      ROWS_SAMPLED PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH
-------------------- --------------------  ------------ ------------------- ------------------- ----------------------
                   0                    0            0                   0                   0                      0
                   0                    0            0                   0                   0                      0
...
               13312                 5312           35                  65                  84                    100
                5760                 4248           45                  76                  79                     98
...
```

Example 2: Determine the dictionary building action and time of dictionary creation for all tables.

```
SELECT TABSCHEMA, TABNAME, DBPARTITIONNUM, DATA_PARTITION_ID, DICT_BUILDER, DICT_BUILD_TIMESTAMP
FROM SYSIBMADM.ADMINTABCOMPRESSINFO
```

The following is an example of output from this query:

```
TABSCHEMA       TABNAME         DBPARTITIONNUM DATA_PARTITION_ID DICT_BUILDER               DICT_BUILD_TIMESTAMP
--------------- --------------- -------------- ----------------- ------------------------- --------------------------
SYSIBM          SYSTABLES                    0                 0 NOT BUILT                 -
SYSIBM          SYSCOLUMNS                   0                 0 NOT BUILT                 -
...
SIMAP2          STAFF                        0                 0 REORG                     2006-08-27-19.07.36.000000
SIMAP2          SALES                        0                 0 NOT BUILT                 -
SIMAP2          CATALOG                      0                 0 NOT BUILT                 -
...

  156 record(s) selected.
```

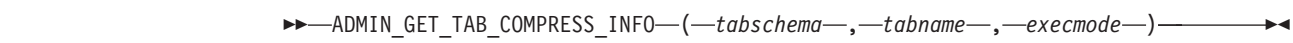
ADMIN_GET_TAB_COMPRESS_INFO table function

The ADMIN_GET_TAB_COMPRESS_INFO table function returns the same information as the ADMINTABCOMPRESSINFO administrative view, but allows you to specify a schema, table name and an execution mode.

Refer to the ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO table function metadata table for a complete list of information that can be returned.

Note: This table function has been deprecated and replaced by the "ADMINTABCOMPRESSINFO administrative view and ADMIN_GET_TAB_COMPRESS_INFO_V97 table function - returns compressed information" on page 228.

Syntax



The schema is SYSPROC.

Table function parameters

tabschema
An input argument of type VARCHAR(128) that specifies a schema name.

tabname
An input argument of type VARCHAR(128) that specifies a table name, a materialized query table name or a hierarchy table name.

execmode
An input argument of type VARCHAR(30) that specifies the execution mode. The execution mode can be one of the following:

- 'REPORT' -- Reports compression information as of last generation. This is the default value.
- 'ESTIMATE' -- Generates new compression information based on the current table.

Authorization

EXECUTE privilege on the ADMIN_GET_TAB_COMPRESS_INFO function.

Examples

Example 1: Retrieve existing compression information for table SIMAP2.STAFF

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('SIMAP2', 'STAFF', 'REPORT'))
AS T
```

The following is an example from output of this query:

```
TABSCHEMA      TABNAME     DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER     DICT_BUILD_TIMESTAMP
---------...- -------...- -------------- ----------------- ------------- ------------...- --------------------------
SIMAP2         STAFF                   0                 0 Y             REORG            2006-08-27-19.07.36.000000

  1 record(s) selected.
```

Output from this query (continued):

```
COMPRESS_DICT_SIZE   EXPAND_DICT_SIZE     ROWS_SAMPLED PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH
-------------------- -------------------- ------------ ------------------- ------------------- ----------------------
               13312                 5312           35                  65                  84                    100
```

Example 2: Retrieve estimated compression information for table SIMAP2.STAFF as of now.

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('SIMAP2', 'STAFF', 'ESTIMATE'))
AS T
```

The following is an example from output of this query:

```
TABSCHEMA      TABNAME     DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER        DICT_BUILD_TIMESTAMP
---------...- -------...- -------------- ----------------- ------------- ----------------...- --------------------------
SIMAP2         STAFF                   0                 0 Y             TABLE FUNCTION      2006-08-28-19.18.13.000000

  1 record(s) selected.
```

Output from this query (continued):

```
COMPRESS_DICT_SIZE   EXPAND_DICT_SIZE     ROWS_SAMPLED PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH
-------------------- -------------------- ------------ -------------------- -------------------- ----------------------
               13508                 6314           68                   72                   89                     98
```

Example 3: Determine the total dictionary size for all tables in the schema SIMAP2

```
SELECT TABSCHEMA, TABNAME, DICT_BUILDER,
 (COMPRESS_DICT_SIZE+EXPAND_DICT_SIZE) AS TOTAL_DICT_SIZE,
 DBPARTITIONNUM, DATA_PARTITION_ID
FROM TABLE (SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('SIMAP2', '', 'REPORT')) AS T
```

Output from this query:

```
TABSCHEMA       TABNAME         DICT_BUILDER                 TOTAL_DICT_SIZE      DBPARTITIONNUM DATA_PARTITION_ID
--------------- --------------- ---------------------------- -------------------- -------------- -----------------
SIMAP2          ACT             NOT BUILT                                     0              0                 0
SIMAP2          ADEFUSR         NOT BUILT                                     0              0                 0
...
SIMAP2          INVENTORY       NOT BUILT                                     0              0                 0
SIMAP2          ORG             NOT BUILT                                     0              0                 0
SIMAP2          PARTTAB         REORG                                     10008              0                 0
SIMAP2          PARTTAB         REORG                                      5464              0                 1
SIMAP2          PARTTAB         REORG                                      8456              0                 2
SIMAP2          PARTTAB         REORG                                      6960              0                 3
SIMAP2          PARTTAB         REORG                                      7136              0                 4
...
SIMAP2          STAFF           REORG                                     18624              0                 0
SIMAP2          SUPPLIERS       NOT BUILT                                     0              0                 0
SIMAP2          TESTTABLE       NOT BUILT                                     0              0                 0

  28 record(s) selected.
```

Example 4: View a report of the dictionary information of tables in the SIMAP2 schema.

```
SELECT * FROM TABLE (SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('SIMAP2', '', 'REPORT'))
AS T
```

Output from this query:

```
TABSCHEMA      TABNAME     DBPARTITIONNUM DATA_PARTITION_ID COMPRESS_ATTR DICT_BUILDER         DICT_BUILD_TIMESTAMP
----------...- -------...- -------------- ----------------- ------------- ----------------...- --------------------------
SIMAP2         T1                      0                 0 Y             NOT BUILT            -
SIMAP2         T2                      0                 0 N             REORG                2007-02-03-17.35.28.000000
SIMAP2         T3                      0                 0 Y             INSPECT              2007-02-03-17.35.44.000000
SIMAP2         T4                      0                 0 N             NOT BUILT            -

  4 record(s) selected.
```

Output from this query (continued):

```
COMPRESS_DICT_SIZE   EXPAND_DICT_SIZE     ROWS_SAMPLED PAGES_SAVED_PERCENT BYTES_SAVED_PERCENT AVG_COMPRESS_REC_LENGTH
-------------------- -------------------- ------------ -------------------- -------------------- ----------------------
                   0                    0            0                    0                    0                      0
                1280                 2562            -                    -                    -                      -
                1340                 2232            -                    -                    -                      -
                   0                    0            0                    0                    0                      0
```

Usage notes

- If both the *tabschema* and *tabname* are specified, information is returned for that specific table only.
- If the *tabschema* is specified but *tabname* is empty (') or NULL, information is returned for all tables in the given schema.
- If the *tabschema* is empty (') or NULL and *tabname* is specified, an error is returned. To retrieve information for a specific table, the table must be identified by both schema and table name.
- If both *tabschema* and *tabname* are empty (') or NULL, information is returned for all tables.

- If *tabschema* or *tabname* do not exist, or *tabname* does not correspond to a table name (type T), a materialized query table name (type S) or a hierarchy table name (type H), an empty result set is returned.
- When the ADMIN_GET_TAB_COMPRESS_INFO table function is retrieving data for a given table, it will acquire a shared lock on the corresponding row of SYSTABLES to ensure consistency of the data that is returned (for example, to ensure that the table is not altered while information is being retrieved for it). The lock will only be held for as long as it takes to retrieve the compression information for the table, and not for the duration of the table function call.

ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO table function metadata

Table 271. ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO table function metadata

Column Name	Data Type	Description
TABSCHEMA	VARCHAR(128)	Schema name
TABNAME	VARCHAR(128)	Table name
DBPARTITIONNUM	SMALLINT	Database partition number
DATA_PARTITION_ID	INTEGER	Data partition number
COMPRESS_ATTR	CHAR(1)	The state of the COMPRESS attribute on the table which can be one of the following: • 'Y' = Row compression is set to yes • 'N' = Row compression is set to no
DICT_BUILDER	VARCHAR(30)	Code path taken to build the dictionary which can be one of the following: • 'INSPECT' = INSPECT ROWCOMPESTIMATE • 'LOAD' = LOAD INSERT/REPLACE • 'NOT BUILT' = no dictionary available • 'REDISTRIBUTE' = REDISTRIBUTE • 'REORG' = REORG RESETDICTIONARY • 'TABLE GROWTH' = INSERT
DICT_BUILD_TIMESTAMP	TIMESTAMP	Timestamp of when the dictionary was built. Timestamp granularity is to the second. If no dictionary is available, then the timestamp is NULL.
COMPRESS_DICT_SIZE	BIGINT	Size of compression dictionary measured in bytes.
EXPAND_DICT_SIZE	BIGINT	Size of the expansion dictionary measured in bytes. If a historical dictionary exists, this value is the sum of the current and historical dictionary sizes.
ROWS_SAMPLED	INTEGER	Number of records that contributed to building the dictionary. Migrated tables with compression dictionaries will return NULL in this column.
PAGES_SAVED_PERCENT	SMALLINT	Percentage of pages saved from compression. This information is based on the record data in the sample buffer only. Migrated tables with compression dictionaries will return NULL in this column.
BYTES_SAVED_PERCENT	SMALLINT	Percentage of bytes saved from compression. This information is based on the record data in the sample buffer only. Migrated tables with compression dictionaries will return NULL in this column.

Table 271. ADMINTABCOMPRESSINFO administrative view and the ADMIN_GET_TAB_COMPRESS_INFO table function metadata (continued)

Column Name	Data Type	Description
AVG_COMPRESS_REC_LENGTH	SMALLINT	Average compressed record length of the records contributing to building the dictionary. Migrated tables with compression dictionaries will return NULL in this column.

GET_DB_CONFIG

Note: This procedure has been deprecated and replaced by the "DBCFG administrative view - Retrieve database configuration parameter information" on page 318.

The schema is SYSPROC.

The GET_DB_CONFIG procedure returns database configuration information. The procedure does not take any arguments.

The procedure returns a single result set with two rows containing a column for each parameter. The first column is named DBCONFIG_TYPE, as shown below.

Table 272. Information returned by the GET_DB_CONFIG procedure

Column name	Data type	Description
DBCONFIG_TYPE	INTEGER	The row with a value of 0 in this column contains the values of the database configuration parameters stored on disk. The row with a value of 1 in this column contains the current values of the database configuration parameters stored in memory.

This procedure requires a user temporary table space that is used to create a global temporary table named DB_CONFIG to store the result set.

Example

Using the command line processor (CLP), change the value of the *logretain* and the *userexit* database configuration parameters. Retrieve the original (on disk) and updated (in memory) values by calling the GET_DB_CONFIG procedure and then querying the resulting global temporary table (DB_CONFIG).

```
CONNECT TO SAMPLE

CREATE BUFFERPOOL MY8KPOOL SIZE 250 PAGESIZE 8K

CREATE USER TEMPORARY TABLESPACE MYTSP2 PAGESIZE
   8K MANAGED BY SYSTEM USING ( 'TSC2' ) BUFFERPOOL MY8KPOOL

UPDATE DB CFG USING LOGRETAIN RECOVERY USEREXIT ON

CALL SYSPROC.GET_DB_CONFIG()
```

```
SELECT DBCONFIG_TYPE, LOGRETAIN, USEREXIT
   FROM SESSION.DB_CONFIG

CONNECT RESET
```

The following is an example of output from this query.

```
DBCONFIG_TYPE LOGRETAIN   USEREXIT
------------- ----------- -----------
            0           1           1
            1           0           0

  2 record(s) selected.
```

GET_DBM_CONFIG

Note: This table function has been deprecated and replaced by the "DBMCFG administrative view - Retrieve database manager configuration parameter information" on page 320.

The schema is SYSFUN.

The GET_DBM_CONFIG table function returns database manager configuration information. The function does not take any arguments.

The function returns a table with two rows containing a column for each parameter. The first column is named DBMCONFIG_TYPE, as shown below.

Table 273. Information returned by the GET_DBM_CONFIG table function

Column name	Data type	Description
DBMCONFIG_TYPE	INTEGER	The row with a value of 0 in this column contains the values of the database manager configuration parameters stored on disk. The row with a value of 1 in this column contains the current values of the database manager configuration parameters stored in memory.

Example

Using the command line processor (CLP), change the value of the *numdb* and the *diaglevel* database manager configuration parameters, and then retrieve the original (on disk) and updated (in memory) values.

```
UPDATE DBM CFG USING NUMDB 32 DIAGLEVEL 4

CONNECT TO SAMPLE

SELECT DBMCONFIG_TYPE, NUMDB, DIAGLEVEL
  FROM TABLE(SYSFUN.GET_DBM_CONFIG()) AS DBMCFG

CONNECT RESET
```

The following is an example of output from this query.

```
DBMCONFIG_TYPE NUMDB       DIAGLEVEL
-------------- ----------- -----------
             0          32           4
             1           8           3

  2 record(s) selected.
```

Health snapshot routines

HEALTH_CONT_HI

The HEALTH_CONT_HI table function returns health indicator information for table space containers from a health snapshot of table spaces in a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_CONT_HI table function.

Example

```
SELECT * FROM TABLE(HEALTH_CONT_HI('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         CONTAINER_NAME                                    ...
-------------------------- ----------------------------------------------...- ...
2006-02-13-12.30.40.759542 D:\DB2\NODE0000\SAMPLE\T0000000\C0000000.CAT    ...
2006-02-13-12.30.40.759542 D:\DB2\NODE0000\SAMPLE\T0000003\C0000000.LRG    ...
2006-02-13-12.30.40.759542 D:\DB2\NODE0000\SAMPLE\T0000004\C0000000.UTM    ...
```

```
2006-02-13-12.30.40.759542 D:\DB2\NODE0000\SAMPLE\T0000001\C0000000.TMP    ...
2006-02-13-12.30.40.759542 D:\DB2\NODE0000\SAMPLE\T0000002\C0000000.LRG    ...

  5 record(s) selected.
```

Output from this query (continued).

```
... NODE_NUMBER HI_ID               HI_VALUE HI_TIMESTAMP              ...
... ----------- ------------------- -------- -------------------------- ...
...           -                3001        1 2006-02-13-12.26.26.158000 ...
...           -                3001        1 2006-02-13-12.26.26.158000 ...
...           -                3001        1 2006-02-13-12.26.26.158000 ...
...           -                3001        1 2006-02-13-12.26.26.158000 ...
...           -                3001        1 2006-02-13-12.26.26.158000 ...
```

Output from this query (continued).

```
... HI_ALERT_STATE       HI_ALERT_STATE_DETAIL HI_FORMULA      HI_ADDITIONAL_INFO
... -------------------- --------------------- -----------...- ------------------
...                    1 Normal                1                                -
...                    1 Normal                1                                -
...                    1 Normal                1                                -
...                    1 Normal                1                                -
...                    1 Normal                1                                -
```

Information returned

Table 274. Information returned by the HEALTH_CONT_HI table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
CONTAINER_NAME	VARCHAR(256)	container_name - Container name
NODE_NUMBER	INTEGER	node_number - Node number
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
HI_VALUE	SMALLINT	The value of the health indicator.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_CONT_HI_HIS

Returns health indicator history information for containers from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_CONT_HI_HIS table function.

Example

```
SELECT * FROM TABLE(HEALTH_CONT_HI_HIS('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         CONTAINER_NAME                                ...
-------------------------- ----------------------------------------...- ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000000\C0000000.CAT ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000000\C0000000.CAT ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000003\C0000000.LRG ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000003\C0000000.LRG ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000004\C0000000.UTM ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000004\C0000000.UTM ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000001\C0000000.TMP ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000001\C0000000.TMP ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000002\C0000000.LRG ...
2006-02-13-12.30.41.915646 D:\DB2\NODE0000\SAMPLE\T0000002\C0000000.LRG ...

  10 record(s) selected.
```

Output from this query (continued).

```
... NODE_NUMBER HI_ID     HI_TIMESTAMP               HI_VALUE HI_ALERT_STATE ...
... ----------- -----...- -------------------------- -------- -------------- ...
...           -      3001 2006-02-13-12.16.25.911000        1              1 ...
...           -      3001 2006-02-13-12.06.26.168000        1              1 ...
...           -      3001 2006-02-13-12.16.25.911000        1              1 ...
...           -      3001 2006-02-13-12.06.26.168000        1              1 ...
```

```
...          -      3001 2006-02-13-12.16.25.911000        1              1 ...
...          -      3001 2006-02-13-12.06.26.168000        1              1 ...
...          -      3001 2006-02-13-12.16.25.911000        1              1 ...
...          -      3001 2006-02-13-12.06.26.168000        1              1 ...
...          -      3001 2006-02-13-12.16.25.911000        1              1 ...
...          -      3001 2006-02-13-12.06.26.168000        1              1 ...
```

Output from this query (continued).

```
... HI_ALERT_STATE_DETAIL HI_FORMULA      HI_ADDITIONAL_INFO
... -------------------- -----------...- ------------------
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
... Normal               1                               -
```

Information returned

Table 275. Information returned by the HEALTH_CONT_HI_HIS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
CONTAINER_NAME	VARCHAR(256)	**container_name** - Container name
NODE_NUMBER	INTEGER	**node_number** - Node number
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_VALUE	SMALLINT	The value of the health indicator.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_CONT_INFO

The HEALTH_CONT_INFO table function returns container information from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_CONT_INFO table function.

Example

```
SELECT * FROM TABLE(HEALTH_CONT_INFO('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         CONTAINER_NAME                                  ...
-------------------------- ----------------------------------------...- ...
2006-02-13-12.30.40.541209 D:\DB2\NODE0000\SAMPLE\T0000000\C0000000.CAT  ...
2006-02-13-12.30.40.541209 D:\DB2\NODE0000\SAMPLE\T0000003\C0000000.LRG  ...
2006-02-13-12.30.40.541209 D:\DB2\NODE0000\SAMPLE\T0000004\C0000000.UTM  ...
2006-02-13-12.30.40.541209 D:\DB2\NODE0000\SAMPLE\T0000001\C0000000.TMP  ...
2006-02-13-12.30.40.541209 D:\DB2\NODE0000\SAMPLE\T0000002\C0000000.LRG  ...

  5 record(s) selected.
```

Output from this query (continued).

```
... TABLESPACE_NAME     NODE_NUMBER ...
... ---------------...- ----------- ...
... SYSCATSPACE                   - ...
... SYSTOOLSPACE                  - ...
... SYSTOOLSTMPSPACE              - ...
... TEMPSPACE1                    - ...
... USERSPACE1                    - ...
```

Output from this query (continued).

```
... ROLLED_UP_ALERT_STATE ROLLED_UP_ALERT_STATE_DETAIL
... -------------------- --------------------------
...                    1 Normal
...                    1 Normal
...                    1 Normal
...                    1 Normal
...                    1 Normal
```

Information returned

Table 276. Information returned by the HEALTH_CONT_INFO table function

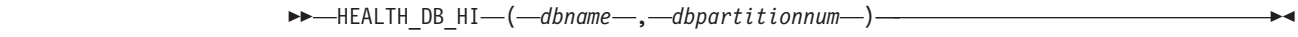
Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
CONTAINER_NAME	VARCHAR(256)	container_name - Container name
TABLESPACE_NAME	VARCHAR(128)	tablespace_name - Table space name
NODE_NUMBER	INTEGER	node_number - Node number
ROLLED_UP_ALERT_STATE	BIGINT	The most severe alert state captured by this snapshot.
ROLLED_UP_ALERT_ STATE_DETAIL	VARCHAR(20)	The text description of the ROLLED_UP_ALERT_STATE column.

HEALTH_DB_HI

The HEALTH_DB_HI table function returns health indicator information from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DB_HI table function.

Example

```
SELECT * FROM TABLE(HEALTH_DB_HI('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         HI_ID     DB_NAME     HI_VALUE ...
-------------------------- -----...- -------...- -------- ...
2006-02-13-12.30.23.949888      1001 SAMPLE             0 ...
2006-02-13-12.30.23.949888      1002 SAMPLE             0 ...
2006-02-13-12.30.23.949888      1003 SAMPLE             0 ...
2006-02-13-12.30.23.949888      1005 SAMPLE             6 ...
2006-02-13-12.30.23.949888      1006 SAMPLE            53 ...
2006-02-13-12.30.23.949888      1008 SAMPLE             3 ...
2006-02-13-12.30.23.949888      1010 SAMPLE             0 ...
2006-02-13-12.30.23.949888      1014 SAMPLE            74 ...
2006-02-13-12.30.23.949888      1015 SAMPLE             1 ...
2006-02-13-12.30.23.949888      1018 SAMPLE             1 ...
2006-02-13-12.30.23.949888      1022 SAMPLE             1 ...

  11 record(s) selected.
```

Output from this query (continued).

```
... HI_TIMESTAMP              HI_ALERT_STATE HI_ALERT_STATE_DETAIL ...
... -------------------------- -------------- -------------------- ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.26.26.158000              1 Normal               ...
... 2006-02-13-12.30.25.640000              2 Attention            ...
... 2006-02-13-12.30.25.640000              2 Attention            ...
... 2006-02-13-12.29.25.281000              2 Attention            ...
```

Output from this query (continued).

```
... HI_FORMULA                                ...
... ----------------------------------------...- ...
... 0                                         ...
... ((0 / 5000) * 100)                        ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
... (((0 - 0) / ((118 - 0) + 1)) * 100)       ...
...                                           ...
...                                           ...
```

```
...                                                 ...
...                                                 ...
...                                                 ...
... ((1170384 / (1170384 + 19229616)) * 100)        ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
... ((11155116032 / 21138935808) * 100)             ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
... ((5264 / (50 * 4096)) * 100)                    ...
... ((0 / 5) * 100)                                 ...
... ((4587520 / 6160384) * 100)                     ...
... -                                               ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
... -                                               ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
...                                                 ...
... -                                               ...
...                                                 ...
...                                                 ...
```

Output from this query (continued).

```
... HI_ADDITIONAL_INFO
... ----------------------------------------
... -
... The high watermark for shared sort
...   memory is "57". "99"% of the time
...   the sort heap allocation is less
...   than or equal to "246". The sort
...   heap (sortheap) database
...   configuration parameter is set
...   to "256". The high watermark for
...   private sort memory is "0".
... The sort heap (sortheap) database
...   configuration parameter is set to
...   "256". The high watermark for
...   private sort memory is "57". The
...   high watermark for shared sort
...   memory is "0"
... The following are the related
...   database configuration parameter
...   settings: logprimary is "3",
...   logsecond is "2", and logfilsiz
...   is "1000". The application with
...   the oldest transaction is "712".
... The following are the related
...   database configuration parameter
...   settings: logprimary is "3",
...   logsecond is "2", and logfilsiz
```

```
...   is "1000", blk_log_dsk_ful is
...   "NO", userexit is "NO",
...   logarchmeth1 is "OFF" and
...   logarchmeth2 is "OFF".
... -
... -
... -
... The scope setting in the reorganization
...   policy is "TABSCHEMA NOT LIKE 'SYS%'".
...   Automatic reorganization (AUTO_REORG)
...   for this database is set to "OFF".
...   The longest estimated reorganization
...   time is "N/A".
... The last successful backup was taken
...   at "N/A". The log space consumed since
...   this last backup has been "N/A" 4KB
...   pages. Automation for database backup
...   is set to "OFF". The last automated
...   backup returned with SQLCODE = "N/A".
...   The longest estimated backup time
...   is "N/A".
... The scope is "N\A". Automatic
...   statistics collection (AUTO_RUNSTATS)
...   is set to "OFF".
```

Information returned

Table 277. Information returned by the HEALTH_DB_HI table function

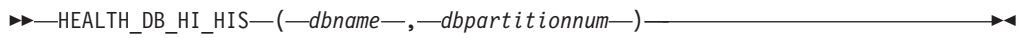
Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
DB_NAME	VARCHAR(128)	db_name - Database name
HI_VALUE	SMALLINT	The value of the health indicator.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_DB_HI_HIS

The HEALTH_DB_HI_HIS table function returns health indicator history information from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DB_HI_HIS table function.

Example

```
SELECT * FROM TABLE(HEALTH_DB_HI_HIS('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         HI_ID     DB_NAME     HI_VALUE ...
-------------------------- -----...- --------...- -------- ...
2006-02-13-12.30.26.325627      1001 SAMPLE             0 ...
...                                                       ...
2006-02-13-12.30.26.325627      1002 SAMPLE             0 ...
...                                                       ...
2006-02-13-12.30.26.325627      1003 SAMPLE             0 ...
...                                                       ...
2006-02-13-12.30.26.325627      1005 SAMPLE             3 ...
...                                                       ...
2006-02-13-12.30.26.325627      1008 SAMPLE             2 ...
...                                                       ...
2006-02-13-12.30.26.325627      1010 SAMPLE             0 ...
...                                                       ...
2006-02-13-12.30.26.325627      1014 SAMPLE            73 ...
...                                                       ...
2006-02-13-12.30.26.325627      1015 SAMPLE             1 ...
...                                                       ...
2006-02-13-12.30.26.325627      1018 SAMPLE             1 ...
...                                                       ...
2006-02-13-12.30.26.325627      1022 SAMPLE             1 ...
...
```

Output from this query (continued).

```
... HI_TIMESTAMP               HI_ALERT_STATE       HI_ALERT_STATE_DETAIL ...
... -------------------------- -------------------- --------------------- ...
... 2006-02-13-12.21.25.649000                    1 Normal                ...
... ...                                                                   ...
... 2006-02-13-12.21.25.649000                    1 Normal                ...
... ...                                                                   ...
... 2006-02-13-12.20.25.182000                    1 Normal                ...
... ...                                                                   ...
... 2006-02-13-12.16.25.911000                    1 Normal                ...
... ...                                                                   ...
... 2006-02-13-12.16.25.911000                    1 Normal                ...
... ...                                                                   ...
... 2006-02-13-12.16.25.911000                    1 Normal                ...
... ...                                                                   ...
... 2006-02-13-12.21.25.649000                    1 Normal                ...
... ...                                                                   ...
... 2006-02-13-12.29.55.461000                    2 Attention             ...
... ...                                                                   ...
... 2006-02-13-12.29.25.281000                    2 Attention             ...
... ...                                                                   ...
... 2006-02-13-12.27.55.743000                    2 Attention             ...
... ...
```

Output from this query (continued).

```
... HI_FORMULA                                ...
... ------------------------------------------ ...
... 0                                         ...
... ...                                       ...
... ((0 / 5000) * 100)                        ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
... ...                                       ...
... (((0 - 0) / ((68 - 0) + 1)) * 100)        ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
... ...                                       ...
... ((698410 / (698410 + 19701590)) * 100)    ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
... ...                                       ...
... ((3920 / (50 * 4096)) * 100)              ...
... ...                                       ...
... ((0 / 4) * 100)                           ...
... ...                                       ...
... ((4521984 / 6160384) * 100)               ...
... ...                                       ...
... -                                         ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
...                                           ...
```

```
... ...                                          ...
... -                                            ...
...                                              ...
...                                              ...
...                                              ...
...                                              ...
...                                              ...
...                                              ...
...                                              ...
... ...                                          ...
... -                                            ...
...                                              ...
...                                              ...
... ...                                          ...
```

Output from this query (continued).

```
... HI_ADDITIONAL_INFO
... ------------------------------------
... -
...
... The high watermark for shared sort
...   memory is "15". "99"% of the time
...   the sort heap allocation is less
...   than or equal to "246". The sort
...   heap (sortheap) database
...   configuration parameter is set
...   to "256". The high watermark
...   for private sort memory is "0".
...
... The sort heap (sortheap) database
...   configuration parameter is set
...   to "256". The high watermark for
...   private sort memory is "15". The
...   high watermark for shared sort
...   memory is "0"
...
... The following are the related
...   database configuration parameter
...   settings: logprimary is "3",
...   logsecond is "2", and logfilsiz
...   is "1000". The application with
...   the oldest transaction is "712".
...
... -
...
... -
...
... -
...
... The scope setting in the
...   reorganization policy is
...   "TABSCHEMA NOT LIKE 'SYS%'".
...   Automatic reorganization
...   (AUTO_REORG) for this database
...   is set to "OFF". The longest
...   estimated reorganization time
...   is "N/A".
...
... The last successful backup was taken
...   at "N/A". The log space consumed
...   since this last backup has been
...   "N/A" 4KB pages. Automation for
...   database backup is set to "OFF". The
...   last automated backup returned with
...   SQLCODE = "N/A". The longest
...   estimated backup time is "N/A".
```

```
...
... The scope is "N\A". Automatic
...   statistics collection
...   (AUTO_RUNSTATS) is set to "OFF".
...
```

Information returned

Table 278. Information returned by the HEALTH_DB_HI_HIS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
DB_NAME	VARCHAR(128)	db_name - Database name
HI_VALUE	SMALLINT	The value of the health indicator.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_DB_HIC

The HEALTH_DB_HIC function returns collection health indicator information from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either

"Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DB_HIC table function.

Example

```
SELECT * FROM TABLE(HEALTH_DB_HIC('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         HI_ID     DB_NAME     ...
-------------------------- -----...- -------...- ...
2006-02-13-12.30.33.870959      1015 SAMPLE      ...
2006-02-13-12.30.33.870959      1022 SAMPLE      ...

  2 record(s) selected.
```

Output from this query (continued).

```
... HI_OBJ_NAME                           HI_OBJ_DETAIL     ...
... ----------------------------------...-- -------------...- ...
... "JESSICAE"."EMPLOYEE"                 REORG TABLE       ...
... "SYSIBM"."SYSDATAPARTITIONEXPRESSION" RUNSTATS          ...
```

Output from this query (continued).

```
... HI_OBJ_STATE HI_OBJ_STATE_DETAIL  HI_TIMESTAMP
... ------------ -------------------- --------------------------
...            2 Attention            2006-02-13-12.24.27.000000
...            2 Attention            2006-02-13-12.29.26.000000
```

Information returned

Table 279. Information returned by the HEALTH_DB_HIC table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
DB_NAME	VARCHAR(128)	**db_name** - Database name
HI_OBJ_NAME	VARCHAR(256)	A name that uniquely identifies an object in the collection.
HI_OBJ_DETAIL	VARCHAR(4096)	Text that describes why the object was added to the collection.

Table 279. Information returned by the HEALTH_DB_HIC table function (continued)

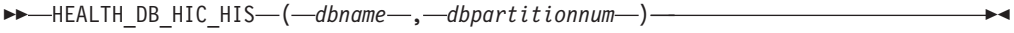
Column name	Data type	Description or corresponding monitor element
HI_OBJ_STATE	BIGINT	The state of the object. Valid states (defined in `sqlmon.h`) include: • NORMAL (1). Action is not required on this object. • ATTENTION (2). Automation is not enabled for this health indicator; action must be taken manually. • AUTOMATED (5). Automation is enabled for this health indicator; action will be started automatically. • AUTOMATE_FAILED (6). Automation is enabled for this health indicator; action was started, but could not complete successfully. Manual intervention is now required.
HI_OBJ_STATE_DETAIL	VARCHAR(20)	A translated string version of the value in the HI_OBJ_STATE column.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.

HEALTH_DB_HIC_HIS

Returns collection health indicator history information from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either

"Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DB_HIC_HIS table function.

Example

```
SELECT * FROM TABLE(HEALTH_DB_HIC_HIS('',-1)) AS T
```

The following is an example of output from this query.

```
HI_HIS_ENTRY_NUM SNAPSHOT_TIMESTAMP         HI_ID                ...
---------------- -------------------------- -------------------- ...
               1 2006-02-13-12.30.34.496720                 1015 ...
               2 2006-02-13-12.30.34.496720                 1022 ...
               3 2006-02-13-12.30.34.496720                 1022 ...
               4 2006-02-13-12.30.34.496720                 1022 ...
               5 2006-02-13-12.30.34.496720                 1022 ...
               6 2006-02-13-12.30.34.496720                 1022 ...
               7 2006-02-13-12.30.34.496720                 1022 ...
               8 2006-02-13-12.30.34.496720                 1022 ...
               9 2006-02-13-12.30.34.496720                 1022 ...
              10 2006-02-13-12.30.34.496720                 1022 ...

  10 record(s) selected.
```

Output from this query (continued).

```
... DB_NAME     HI_OBJ_NAME                              HI_OBJ_STATE ...
... -------...- ---------------------------------...- ------------ ...
... SAMPLE      "JESSICAE"."EMPLOYEE"                               2 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               2 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               2 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               2 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               1 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               1 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               1 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               1 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               1 ...
... SAMPLE      "SYSIBM"."SYSDATAPARTITIONEXPRESSION"               1 ...
```

Output from this query (continued).

```
... HI_OBJ_STATE_DETAIL  HI_TIMESTAMP
... -------------------- --------------------------
... Attention            2006-02-10-09.04.57.000000
... Attention            2006-02-13-12.27.56.000000
... Attention            2006-02-13-12.26.27.000000
... Attention            2006-02-13-12.24.56.000000
... Normal               2006-02-13-12.23.28.000000
... Normal               2006-02-13-12.21.56.000000
... Normal               2006-02-13-12.20.26.000000
... Normal               2006-02-13-12.18.57.000000
... Normal               2006-02-13-12.17.27.000000
... Normal               2006-02-13-12.15.56.000000
```

Information returned

Table 280. Information returned by the HEALTH_DB_HIC_HIS table function

Column name	Data type	Description or corresponding monitor element
HI_HIS_ENTRY_NUM	INTEGER	A number that uniquely identifies the history entry.
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
DB_NAME	VARCHAR(128)	**db_name** - Database name
HI_OBJ_NAME	VARCHAR(256)	A name that uniquely identifies an object in the collection.
HI_OBJ_STATE	BIGINT	The state of the object. Valid states (defined in `sqlmon.h`) include: • NORMAL (1). Action is not required on this object. • ATTENTION (2). Automation is not enabled for this health indicator; action must be taken manually. • AUTOMATED (5). Automation is enabled for this health indicator; action will be started automatically. • AUTOMATE_FAILED (6). Automation is enabled for this health indicator; action was started, but could not complete successfully. Manual intervention is now required.
HI_OBJ_STATE_DETAIL	VARCHAR(20)	A translated string version of the value in the HI_OBJ_STATE column.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.

HEALTH_DB_INFO

The HEALTH_DB_INFO table function returns information from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DB_INFO table function.

Example

```
SELECT * FROM TABLE(HEALTH_DB_INFO('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         DB_NAME     INPUT_DB_ALIAS      ...
-------------------------- -------...- ---------------...- ...
2006-02-13-12.30.23.340081 SAMPLE      SAMPLE              ...

  1 record(s) selected.
```

Output from this query (continued).

```
... DB_PATH                      DB_LOCATION SERVER_PLATFORM ...
... -------------------------...- ----------- --------------- ...
... D:\DB2\NODE0000\SQL00003\              1               5 ...
```

Output from this query (continued).

```
... ROLLED_UP_ALERT_STATE ROLLED_UP_ALERT_STATE_DETAIL
... --------------------- ----------------------------
...                     4 Alarm
```

Information returned

Table 281. Information returned by the HEALTH_DB_INFO table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.

Table 281. Information returned by the HEALTH_DB_INFO table function (continued)

Column name	Data type	Description or corresponding monitor element
DB_NAME	VARCHAR(128)	db_name - Database name
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
DB_PATH	VARCHAR(1024)	db_path - Database path
DB_LOCATION	INTEGER	db_location - Database location
SERVER_PLATFORM	INTEGER	server_platform - Server operating system
ROLLED_UP_ALERT_STATE	BIGINT	The most severe alert state captured by this snapshot.
ROLLED_UP_ALERT_ STATE_DETAIL	VARCHAR(20)	The text description of the ROLLED_UP_ALERT_STATE column.

HEALTH_DBM_HI

The HEALTH_DBM_HI table function returns health indicator information from a health snapshot of the DB2 database manager.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameter

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DBM_HI table function.

Example

```
SELECT * FROM TABLE(HEALTH_DBM_HI(-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         HI_ID     SERVER_INSTANCE_NAME     ...
-------------------------- -----...- --------------------...- ...
2006-02-13-12.30.19.773632         1 DB2                      ...
2006-02-13-12.30.19.773632         4 DB2                      ...

  2 record(s) selected.
```

Output from this query (continued).

```
... HI_VALUE HI_TIMESTAMP              HI_ALERT_STATE HI_ALERT_STATE_DETAIL ...
... -------- -------------------------- -------------- -------------------- ...
...        0 2006-02-13-12.26.26.158000              1 Normal               ...
...      100 2006-02-13-12.26.26.158000              4 Alarm                ...
```

Output from this query (continued).

```
... HI_FORMULA                HI_ADDITIONAL_INFO
... --------------------...-  ------------------
... 0                                           -
... ((327680 / 327680) * 100)                   -
```

Table 282. Information returned by the HEALTH_DBM_HI table function

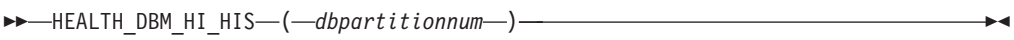
Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
SERVER_INSTANCE_NAME	VARCHAR(128)	server_instance_name - Server instance name
HI_VALUE	SMALLINT	The value of the health indicator.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_DBM_HI_HIS

The HEALTH_DBM_HI_HIS table function returns health indicator history information from a health snapshot of the DB2 database manager.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DBM_HI_HIS table function.

Example

```
SELECT * FROM TABLE(HEALTH_DBM_HI_HIS(-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         HI_ID     SERVER_INSTANCE_NAME     HI_VALUE ...
-------------------------- -----...- --------------------...- -------- ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         1 DB2                             0 ...
2006-02-13-12.30.20.460905         4 DB2                           100 ...
2006-02-13-12.30.20.460905         4 DB2                           100 ...
2006-02-13-12.30.20.460905         4 DB2                           100 ...
2006-02-13-12.30.20.460905         4 DB2                           100 ...
2006-02-13-12.30.20.460905         4 DB2                            60 ...
2006-02-13-12.30.20.460905         4 DB2                            60 ...
2006-02-13-12.30.20.460905         4 DB2                            60 ...
2006-02-13-12.30.20.460905         4 DB2                            60 ...
2006-02-13-12.30.20.460905         4 DB2                            60 ...

  18 record(s) selected.
```

Output for this query (continued).

```
... HI_TIMESTAMP              HI_ALERT_STATE HI_ALERT_STATE_DETAIL ...
... -------------------------- -------------- -------------------- ...
... 2006-02-13-12.21.25.649000              1 Normal               ...
... 2006-02-13-12.16.25.911000              1 Normal               ...
... 2006-02-13-12.11.25.377000              1 Normal               ...
... 2006-02-13-12.06.26.168000              1 Normal               ...
... 2006-02-13-12.01.25.165000              1 Normal               ...
... 2006-02-13-11.56.25.927000              1 Normal               ...
... 2006-02-13-11.51.25.452000              1 Normal               ...
... 2006-02-13-11.46.25.211000              1 Normal               ...
... 2006-02-13-11.41.25.972000              1 Normal               ...
... 2006-02-13-12.21.25.649000              4 Alarm                ...
... 2006-02-13-12.16.25.911000              4 Alarm                ...
```

```
... 2006-02-13-12.11.25.377000             4 Alarm                ...
... 2006-02-13-12.06.26.168000             4 Alarm                ...
... 2006-02-13-12.01.25.165000             1 Normal               ...
... 2006-02-13-11.56.25.927000             1 Normal               ...
... 2006-02-13-11.51.25.452000             1 Normal               ...
... 2006-02-13-11.46.25.211000             1 Normal               ...
... 2006-02-13-11.41.25.972000             1 Normal               ...
```

Output for this query (continued).

```
... HI_FORMULA                   HI_ADDITIONAL_INFO
... -------------------------...- ------------------
... 0                                               -
... 0                                               -
... 0                                               -
... 0                                               -
... 0                                               -
... 0                                               -
... 0                                               -
... 0                                               -
... 0                                               -
... ((327680 / 327680) * 100)                       -
... ((327680 / 327680) * 100)                       -
... ((327680 / 327680) * 100)                       -
... ((327680 / 327680) * 100)                       -
... ((196608 / 327680) * 100)                       -
... ((196608 / 327680) * 100)                       -
... ((196608 / 327680) * 100)                       -
... ((196608 / 327680) * 100)                       -
... ((196608 / 327680) * 100)                       -
```

Information returned

Table 283. Information returned by the HEALTH_DBM_HI_HIS table function

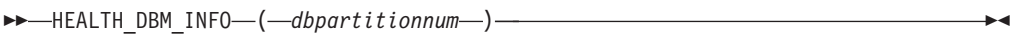
Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
SERVER_INSTANCE_NAME	VARCHAR(128)	server_instance_name - Server instance name
HI_VALUE	SMALLINT	The value of the health indicator.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_DBM_INFO

The HEALTH_DBM_INFO function returns information from a health snapshot of the DB2 database manager.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_DBM_INFO table function.

Example

```
SELECT * FROM TABLE(HEALTH_DBM_INFO(-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         SERVER_INSTANCE_NAME       ROLLED_UP_ALERT_STATE  ...
-------------------------- --------------------...--- ---------------------  ...
2006-02-13-12.30.19.663924 DB2                                           4  ...

  1 record(s) selected.
```

Output from this query (continued).

```
... ROLLED_UP_ALERT_STATE_DETAIL DB2START_TIME              ...
... ---------------------------- -------------------------- ...
... Alarm                        2006-02-09-10.56.18.126182 ...
```

Output from this query (continued).

```
... LAST_RESET        NUM_NODES_IN_DB2_INSTANCE
... ----------...--- ------------------------
... -                                       1
```

Information returned

Table 284. Information returned by the HEALTH_DBM_INFO table function

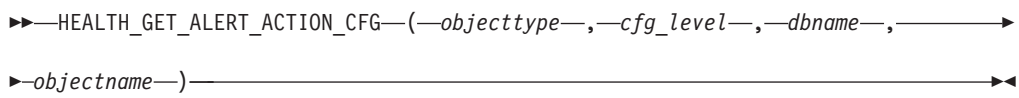
Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
SERVER_INSTANCE_NAME	VARCHAR(128)	server_instance_name - Server instance name
ROLLED_UP_ALERT_STATE	BIGINT	The most severe alert state captured by this snapshot.
ROLLED_UP_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the ROLLED_UP_ALERT_STATE column.
DB2START_TIME	TIMESTAMP	db2start_time - Start database manager timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
NUM_NODES_IN_DB2_INSTANCE	INTEGER	num_nodes_in_db2_instance - Number of nodes in database partition

HEALTH_GET_ALERT_ACTION_CFG

Returns health alert action configuration settings for various object types (database manager, database, table space, and table space container) and for various configuration levels (install default, instance, global, and object).

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameters

objecttype
An input argument of type VARCHAR(3) that indicates the object type. The value must be one of the following case-insensitive values:

- 'DBM' for database manager
- 'DB' for database
- 'TS' for table space
- 'TSC' for table space container

Note: Leading and trailing spaces will be ignored.

cfg_level
An input argument of type VARCHAR(1) that indicates the configuration level. The value must be one of the following case-insensitive values:

- For *objecttype* 'DBM': 'D' for install default; 'G' or 'O' for instance level.
- For *objecttype* that is not 'DBM': 'D' for install default; 'G' for global level; 'O' for object level.

dbname
An input argument of type VARCHAR(128) that indicates the database name. The database name must be provided if *objecttype* is 'DB', 'TS', or 'TSC', and *cfg_level* is 'O'. For all other combinations of *objecttype* and *cfg_level*, the *dbname* parameter should be NULL (or an empty string).

objectname
An input argument of type VARCHAR(1024) that indicates the object name, for example, <table space name> or <table space name>.<container name>. The object name must be provided if *objecttype* is 'TS' or 'TSC', and *cfg_level* is 'O'. For all other combinations of *objecttype* and *cfg_level*, the *objectname* parameter should be NULL (or an empty string).

Authorization

EXECUTE privilege on the HEALTH_GET_ALERT_ACTION_CFG table function.

Examples

Example 1: Retrieve object level alert action configuration settings for database SAMPLE for health indicator ID 1004.

```
SELECT OBJECTTYPE, CFG_LEVEL, SUBSTR(DBNAME,1,8) AS DBNAME,
   SUBSTR(OBJECTNAME,1,8) AS OBJECTNAME, ID, IS_DEFAULT,
   SUBSTR(CONDITION,1,10) AS CONDITION, ACTIONTYPE,
   SUBSTR(ACTIONNAME,1,30) AS ACTIONNAME, SUBSTR(USERID,1,8) AS USERID,
   SUBSTR(HOSTNAME,1,10) AS HOSTNAME, SCRIPT_TYPE,
   SUBSTR(WORKING_DIR,1,10) AS WORKING_DIR, TERMINATION_CHAR,
   SUBSTR(PARAMETERS,1,10) AS PARAMETERS
FROM TABLE(HEALTH_GET_ALERT_ACTION_CFG('DB','O','SAMPLE','')) AS ACTION_CFG
WHERE ID = 1004
```

The following is an example of output for this query.

```
OBJECTTYPE CFG_LEVEL DBNAME   OBJECTNAME ID        IS_DEFAULT CONDITION
---------- --------- -------- ---------- --...---- ---------- ----------
DB         O         SAMPLE                  1004          1 ALARM
DB         O         SAMPLE                  1004          1 ALARM

  2 record(s) selected.
```

Output for this query (continued).

```
... ACTIONTYPE ACTIONNAME                     USERID   HOSTNAME
... ---------- ------------------------------ -------- ----------
... S          ~/health_center/script/scrpn6  uid1     -
... T          00.0005                        uid1     HOST3
```

Output for this query (continued).

```
... SCRIPT_TYPE WORKING_DIR TERMINATION_CHAR PARAMETERS
... ----------- ----------- ---------------- ----------
... 0           ~/health_c  -                -
... -           -           -                -
```

Example 2: Retrieve the condition, action type, action name, hostname, and script type for database SAMPLE for health indicator ID 1004.

```
SELECT CONDITION, ACTIONTYPE, SUBSTR(ACTIONNAME,1,35) AS ACTIONNAME,
   SUBSTR(USERID,1,8) AS USERID, SUBSTR(HOSTNAME,1,10) AS HOSTNAME, SCRIPT_TYPE
FROM TABLE(HEALTH_GET_ALERT_ACTION_CFG('DB','O','SAMPLE','')) AS ALERT_ACTION_CFG
WHERE ID=1004
```

The following is an example of output for this query.

```
CONDITION       ACTIONTYPE ACTIONNAME                          ...
---------...--- ---------- ------------------------------- ...
ALARM           S          ~/health_center/script/scrpn6    ...
ALARM           T          00.0005                          ...

  2 record(s) selected.
```

Output for this query (continued).

```
... USERID   HOSTNAME   SCRIPT_TYPE
... -------- ---------- -----------
... uid1     -          0
... uid1     HOST3      -
```

Usage notes

The HEALTH_GET_IND_DEFINITION table function can be used to map health indicator IDs to the health indicator names.

Information returned

Table 285. Information returned by the HEALTH_GET_ALERT_ACTION_CFG table function

Column name	Data type	Description
OBJECTTYPE	VARCHAR(3)	Object type.
CFG_LEVEL	CHAR(1)	Configuration level.
DBNAME	VARCHAR(128)	Database name.
OBJECTNAME	VARCHAR(512)	Object name.
ID	BIGINT	Health indicator ID.
IS_DEFAULT	SMALLINT	Whether the settings is the default: 1 if it is the default, 0 if it is not the default, Null if it is not applicable.
CONDITION	VARCHAR(512)	Alert condition upon which the action is triggered.
ACTIONTYPE	CHAR(1)	Action type: 'S' for script action or 'T' for task action.
ACTIONNAME	VARCHAR(5000)	If ACTIONTYPE is 'S', this is the script path name. If ACTIONTYPE is 'T', this is the task ID.
USERID	VARCHAR(1024)	User name under which the action will be executed.
HOSTNAME	VARCHAR(255)	Host system name.

Table 285. Information returned by the HEALTH_GET_ALERT_ACTION_CFG table function (continued)

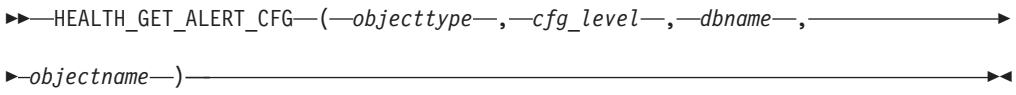
Column name	Data type	Description
SCRIPT_TYPE	CHAR(1)	Script type: If ACTIONTYPE is 'S', 'O' for operating system command script or 'D' for DB2 command script; If ACTIONTYPE is 'T', Null.
WORKING_DIR	VARCHAR(5000)	The working directory for the script if ACTIONTYPE is 'S' or Null if ACTIONTYPE is 'T'.
TERMINATION_CHAR	VARCHAR(4)	The statement termination character if it is a DB2 command script action, otherwise Null.
PARAMETERS	VARCHAR(200)	The command line parameters if it is an operating system command script action.

HEALTH_GET_ALERT_CFG

Returns health alert configuration settings for various object types (database manager, database, table space, table space container) and for various configuration levels (install default, global, and object).

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameters

objecttype
An input argument of type VARCHAR(3) that indicates the object type. The value must be one of the following case-insensitive values:

- 'DBM' for database manager
- 'DB' for database
- 'TS' for table space
- 'TSC' for table space container

Note: Leading and trailing spaces will be ignored.

cfg_level
An input argument of type VARCHAR(1) that indicates the configuration level. The value must be one of the following case-insensitive values:

- For *objecttype* 'DBM': 'D' for install default; 'G' or 'O' for instance level.
- For *objecttype* that is not 'DBM': 'D' for install default; 'G' for global level; 'O' for object level.

dbname
An input argument of type VARCHAR(128) that indicates the database name. The database name must be provided if *objecttype* is 'DB', 'TS', or 'TSC', and *cfg_level* is 'O'. For all other combinations of *objecttype* and *cfg_level*, the *dbname* parameter should be NULL (or an empty string).

objectname
An input argument of type VARCHAR(1024) that indicates the object name, for example, <table space name> or <table space name>.<container name>. The object name must be provided if *objecttype* is 'TS' or 'TSC', and *cfg_level* is 'O'. For all other combinations of *objecttype* and *cfg_level*, the *objectname* parameter should be NULL (or an empty string).

Authorization

EXECUTE privilege on the HEALTH_GET_ALERT_CFG table function.

Examples

Example 1: Retrieve the object level alert configuration settings for database SAMPLE.

```
SELECT * FROM TABLE(SYSPROC.HEALTH_GET_ALERT_CFG('DB','O','SAMPLE',''))
   AS ALERT_CFG
```

The following is an example of output for this query.

```
OBJECTTYPE CFG_LEVEL DBNAME           OBJECTNAME             ...
---------- --------- --------...----- -------------...------ ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
DB         O         SAMPLE                                 ...
...                                                         ...
```

Output for this query (continued).

```
... ID                   IS_DEFAULT WARNING_THRESHOLD     ...
... -------------------- ---------- --------------------  ...
...                 1001          0                    0  ...
...                 1018          0                    0  ...
...                 1015          0                    0  ...
...                 1022          0                    0  ...
...                 1002          1                   95  ...
...                 1003          1                   30  ...
...                 1004          1                   60  ...
```

```
...                 1005         1                  75  ...
...                 1006         1                  75  ...
...                 1007         1                   5  ...
...                 1008         1                  75  ...
...                 1009         1                   5  ...
...                 1010         1                  50  ...
...                 1011         1                  80  ...
```

Output for this query (continued).

```
... ALARM_THRESHOLD      SENSITIVITY          EVALUATE ACTION_ENABLED
... -------------------- -------------------- -------- --------------
...                   0                    0        0              0
...                   0                    0        1              0
...                   0                    0        1              0
...                   0                    0        1              0
...                 100                    0        0              0
...                  50                    0        1              0
...                  30                    0        1              0
...                  85                    0        1              0
...                  85                    0        1              0
...                  10                    0        1              0
...                  85                    0        1              0
...                  10                    0        1              0
...                  70                    0        1              0
...                  70                    0        0              0
```

Example 2: Retrieve the warning and alarm thresholds for the health indicator ID '2002' for table space USERSPACE1 in database SAMPLE.

```
SELECT WARNING_THRESHOLD, ALARM_THRESHOLD
   FROM TABLE(SYSPROC.HEALTH_GET_ALERT_CFG('TS','O','SAMPLE','USERSPACE1'))
   AS T WHERE ID = 2002
```

The following is an example of output for this query.

```
WARNING_THRESHOLD    ALARM_THRESHOLD
-------------------- --------------------
                  80                   90
SQL22004N  Cannot find the requested configuration for the given object.
Returning default configuration for "tablespaces".


  1 record(s) selected with 1 warning messages printed.
```

Usage notes

The HEALTH_GET_IND_DEFINITION table function can be used to map health indicator IDs to the health indicator names.

Example: Retrieve the warning and alarm thresholds for the health indicator Tablespace Utilization (ts.ts_util) for table space USERSPACE1 in database SAMPLE.

```
WITH HINAME(ID) AS (SELECT ID FROM TABLE(SYSPROC.HEALTH_GET_IND_DEFINITION('')) AS W
   WHERE NAME = 'ts.ts_util')
SELECT WARNING_THRESHOLD, ALARM_THRESHOLD
   FROM TABLE(SYSPROC.HEALTH_GET_ALERT_CFG('TS','O','SAMPLE','USERSPACE1')) AS T,
   HINAME AS H
   WHERE T.ID = H.ID
```

The following is an example of output for this query.

```
WARNING_THRESHOLD    ALARM_THRESHOLD
-------------------- --------------------
                  80                   90
SQL22004N  Cannot find the requested configuration for the given object.
```

```
Returning default configuration for "tablespaces".

  1 record(s) selected with 1 warning messages printed.
```

Information returned

Table 286. Information returned by the HEALTH_GET_ALERT_CFG table function

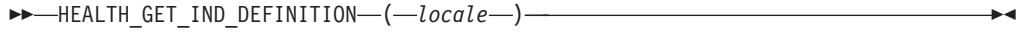
Column name	Data type	Description
OBJECTTYPE	VARCHAR(3)	Object type.
CFG_LEVEL	VARCHAR(1)	Configuration level.
DBNAME	VARCHAR(128)	Database name.
OBJECTNAME	VARCHAR(512)	Object name.
ID	BIGINT	Health indicator ID.
IS_DEFAULT	SMALLINT	Whether the settings is the default: 1 if it is the default, 0 if it is not the default or Null if not applicable.
WARNING_THRESHOLD	BIGINT	Warning threshold. Null if not applicable.
ALARM_THRESHOLD	BIGINT	Alarm threshold. Null if not applicable.
SENSITIVITY	BIGINT	Health indicator sensitivity.
EVALUATE	SMALLINT	1 if this health indicator is being evaluated or 0 if it is not being evaluated.
ACTION_ENABLED	SMALLINT	1 if an action is enabled to run upon an alert occurrence or 0 if no action is enabled to run upon an alert occurrence.

HEALTH_GET_IND_DEFINITION

Returns the health indicator definitions.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax



The schema is SYSPROC.

Table function parameter

locale
An input argument of type VARCHAR(33) that indicates the locale in which the translatable output is to be returned. If the input locale is not supported by the database server, an SQL warning message is issued, and the default

language (English) is used. If the input locale is not provided, that is, its value is NULL (or an empty string), the default language is used.

Authorization

EXECUTE privilege on the HEALTH_GET_IND_DEFINITION table function.

Examples

Example 1: Retrieve the type and short description for health indicator db.db_op_status in French.

```
SELECT TYPE, SHORT_DESCRIPTION
   FROM TABLE(SYSPROC.HEALTH_GET_IND_DEFINITION('fr_FR'))
   AS IND_DEFINITION WHERE NAME = 'db.db_op_status'
```

The following is an example of output for this query.

```
TYPE            SHORT_DESCRIPTION
--------------- ----------------------------------------...---------
STATE           Etat opérationnel de la base de données

  1 record(s) selected.
```

Example 2: Retrieve the short description for health indicator ID 1001 in English.

```
SELECT SHORT_DESCRIPTION FROM TABLE(SYSPROC.HEALTH_GET_IND_DEFINITION('en_US')
   AS IND_DEFINITION WHERE ID = 1001
```

The following is an example of output for this query.

```
SHORT_DESCRIPTION
----------------------------...-------------
Database Operational State
```

Example 3: Retrieve all health indicator IDs and names.

```
SELECT ID, NAME FROM TABLE(HEALTH_GET_IND_DEFINITION('')) AS T
```

The following is an example of output for this query.

```
ID                   NAME
-------------------- ----------------------------
                   1 db2.db2_op_status
                   2 db2.sort_privmem_util
                   4 db2.mon_heap_util
                1001 db.db_op_status
                1002 db.sort_shrmem_util
...
                2001 ts.ts_op_status
                2002 ts.ts_util
...
                3002 tsc.tscont_util
                1015 db.tb_reorg_req
...
```

Information returned

Table 287. Information returned by the HEALTH_GET_IND_DEFINITION table function

Column name	Data type	Description
ID	BIGINT	Health indicator ID.
NAME	VARCHAR(128)	Health indicator name.

Table 287. Information returned by the HEALTH_GET_IND_DEFINITION table function (continued)

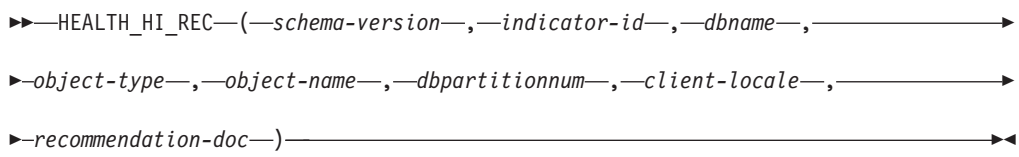
Column name	Data type	Description
SHORT_DESCRIPTION	VARCHAR(1024)	Health indicator short description.
LONG_DESCRIPTION	VARCHAR(32672)	Health indicator long description.
TYPE	VARCHAR(16)	Health indicator type. Possible values are: • 'THRESHOLD_UPPER': upper-bounded threshold-based health indicators. • 'THRESHOLD_LOWER': lower-bounded threshold-based health indicators. • 'STATE': state-based health indicators. • 'COLLECTION_STATE': collection state-based health indicators.
UNIT	VARCHAR(1024)	Unit of the health indicator values and thresholds or Null if not applicable.
CATEGORY	VARCHAR(1024)	Health indicator category.
FORMULA	VARCHAR(512)	Health indicator formula.
REFRESH_INTERVAL	BIGINT	Health indicator evaluation interval in seconds.

HEALTH_HI_REC

Retrieves a set of recommendations that address a health indicator in alert state on a particular DB2 object. Recommendations are returned in an XML document that contains information about actions that can be taken (for example, scripts that can be run) to resolve the alert state.

Important: This procedure has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New in Version 9.7* book.

Syntax



The schema is SYSPROC.

Any scripts that are returned by this procedure must be invoked from the instance on which the health indicator entered the alert state.

If the specified health indicator on the identified object is not in an alert state, an error is returned (SQLSTATE 5U0ZZ).

Procedure parameters

schema-version
An input argument of type INTEGER that specifies the version ID of the schema used to represent the XML document. The recommendation document will only contain elements and attributes that were defined for that schema version. Valid schema versions are defined in `db2ApiDf.h`, located in the include subdirectory of the `sqllib` directory.

indicator-id
An input argument of type INTEGER that specifies the numeric identifier of the health indicator for which recommendations are being requested. Valid health indicator IDs are defined in `sqlmon.h`, located in the include subdirectory of the `sqllib` directory.

dbname
An input argument of type VARCHAR(255) that specifies an alias name for the database against which the health indicator entered an alert state, and when object type is either `DB2HEALTH_OBJTYPE_TS_CONTAINER`, `DB2HEALTH_OBJTYPE_TABLESPACE`, or `DB2HEALTH_OBJTYPE_DATABASE`. Specify `NULL` otherwise.

object-type
An input argument of type INTEGER that specifies the type of object on which the health indicator entered an alert state. Valid object types are defined in `sqlmon.h`, located in the include subdirectory of the sqllib directory.

object-name
An input argument of type VARCHAR(255) that specifies the name of a table space or table space container when the object type is set to `DB2HEALTH_OBJTYPE_TABLESPACE` or `DB2HEALTH_OBJTYPE_TS_CONTAINER`. Specify `NULL` if the object type is `DB2HEALTH_OBJTYPE_DATABASE` or `DB2HEALTH_OBJTYPE_DATABASE_MANAGER`. In the case of a table space container, the object name is specified as *table_space_name.container_name*.

dbpartitionnum
An input argument of type INTEGER that specifies the number of the database partition on which the health indicator entered an alert state. Valid values are `0` to `999`, `-1` (which specifies the currently connected database partition), and `-2` (which specifies all active database partitions). An active database partition is a partition where the database is available for connection and use by applications.

client-locale
An input argument of type VARCHAR(33) that specifies a client language identifier. Use this parameter to specify the language in which recommendations are to be returned. If no value is specified, `'En_US'` (English) will be used. Note that if the message files for the specified locale are not available on the server, `'En_US'` will be used as the default.

recommendation-doc
An output argument of type BLOB(2M) that contains the recommendation document (XML), formatted according to the DB2 Health Recommendation schema definition (see the XML schema `DB2RecommendationSchema.xsd`, located

in the misc subdirectory of the sqllib directory). The XML document is encoded in UTF-8, and text in the document is in the locale of the caller, or English, if messages are not available in the caller's locale at the target instance.

HEALTH_TBS_HI

Returns health indicator information for table spaces from a health snapshot of table spaces in a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_TBS_HI table function.

Example

```
SELECT * FROM TABLE(HEALTH_TBS_HI('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         TABLESPACE_NAME     HI_ID     HI_VALUE ...
-------------------------- ---------------...- -----...- -------- ...
2006-02-13-12.30.35.229196 SYSCATSPACE              2001        0 ...
2006-02-13-12.30.35.229196 SYSCATSPACE              2002       99 ...
2006-02-13-12.30.35.229196 SYSCATSPACE              2003        0 ...
2006-02-13-12.30.35.229196 SYSTOOLSPACE             2001        0 ...
2006-02-13-12.30.35.229196 SYSTOOLSPACE             2002       62 ...
2006-02-13-12.30.35.229196 SYSTOOLSPACE             2003        0 ...
2006-02-13-12.30.35.229196 SYSTOOLSTMPSPACE         2001        0 ...
2006-02-13-12.30.35.229196 TEMPSPACE1               2001        0 ...
2006-02-13-12.30.35.229196 USERSPACE1               2001        0 ...
```

```
2006-02-13-12.30.35.229196 USERSPACE1               2002      100 ...
2006-02-13-12.30.35.229196 USERSPACE1               2003        0 ...

  11 record(s) selected.
```

Output from this query (continued).

```
... HI_TIMESTAMP               HI_ALERT_STATE  HI_ALERT_STATE_DETAIL ...
... -------------------------- --------------- --------------------- ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               4 Alarm                 ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
... 2006-02-13-12.26.26.158000               4 Alarm                 ...
... 2006-02-13-12.26.26.158000               1 Normal                ...
```

Output from this query (continued).

```
... HI_FORMULA                HI_ADDITIONAL_INFO
... ---------------------...- ------------------------------------------------
... 0                         -
... ((9376 / 9468) * 100)     The short term table space growth rate
                                from "02/13/2006 11:26:26.000158" to
                                "02/13/2006 12:26:26.000158" is "N/A"
                                bytes per second and the long term growth
                                rate from "02/12/2006 12:26:26.000158"
                                to "02/13/2006 12:26:26.000158" is "N/A"
                                bytes per second. Time to fullness is
                                projected to be "N/A" and "N/A"
                                respectively. The table space is defined
                                with automatic storage set to "YES" and
                                automatic resize enabled set to "YES".
... 0                         The table space is defined with automatic
                                storage set to "YES" and automatic resize
                                enabled set to "YES". The following are
                                the automatic resize settings: increase
                                size (bytes) "-1", increase size (percent)
                                "N/A", maximum size (bytes) "-1". The
                                current table space size (bytes) is
                                "38797312".
... 0                         -
... ((156 / 252) * 100)       The short term table space growth rate
                                from "02/13/2006 11:26:26.000158" to
                                "02/13/2006 12:26:26.000158" is "N/A"
                                bytes per second and the long term growth
                                rate from "02/12/2006 12:26:26.000158"
                                to "02/13/2006 12:26:26.000158" is "N/A"
                                bytes per second. Time to fullness is
                                projected to be "N/A" and "N/A"
                                respectively. The table space is defined
                                with automatic storage set to "YES" and
                                automatic resize enabled set to "YES".
... 0                         The table space is defined with automatic
                                storage set to "YES" and automatic resize
                                enabled set to "YES". The following are
                                the automatic resize settings: increase
                                size (bytes) "-1", increase size (percent)
                                "N/A", maximum size (bytes) "-1". The
                                current table space size (bytes) is
                                "1048576".
... 0                         -
... 0                         -
... 0                         -
```

```
... ((1504 / 1504) * 100)     The short term table space growth rate from
                                "02/13/2006 11:26:26.000158" to
                                "02/13/2006 12:26:26.000158" is "N/A"
                                bytes per second and the long term growth
                                rate from "02/12/2006 12:26:26.000158" to
                                "02/13/2006 12:26:26.000158" is "N/A" bytes
                                per second. Time to fullness is projected
                                to be "N/A" and "N/A" respectively. The
                                table space is defined with automatic storage
                                set to "YES" and automatic resize enabled
                                set to "YES".
... 0                         The table space is defined with automatic
                                storage set to "YES" and automatic resize
                                enabled set to "YES". The following are
                                the automatic resize settings: increase
                                size (bytes) "-1", increase size (percent)
                                "N/A", maximum size (bytes) "-1". The
                                current table space size (bytes) is
                                "6291456".
```

Information returned

Table 288. Information returned by the HEALTH_TBS_HI table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
HI_VALUE	SMALLINT	The value of the health indicator.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_TBS_HI_HIS

The HEALTH_TBS_HI_HIS table function returns health indicator history information for table spaces from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_TBS_HI_HIS table function.

Example

```
SELECT * FROM TABLE(HEALTH_TBS_HI_HIS('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         TABLESPACE_NAME     HI_ID     ...
-------------------------- ---------------...- -----...- ...
2006-02-13-12.30.37.181478 SYSCATSPACE              2001 ...
2006-02-13-12.30.37.181478 SYSCATSPACE              2001 ...
2006-02-13-12.30.37.181478 SYSCATSPACE              2002 ...
2006-02-13-12.30.37.181478 SYSCATSPACE              2002 ...
2006-02-13-12.30.37.181478 SYSCATSPACE              2003 ...
2006-02-13-12.30.37.181478 SYSCATSPACE              2003 ...
2006-02-13-12.30.37.181478 SYSTOOLSPACE             2001 ...
2006-02-13-12.30.37.181478 SYSTOOLSPACE             2001 ...
2006-02-13-12.30.37.181478 SYSTOOLSPACE             2002 ...
2006-02-13-12.30.37.181478 SYSTOOLSPACE             2002 ...
2006-02-13-12.30.37.181478 SYSTOOLSPACE             2003 ...
2006-02-13-12.30.37.181478 SYSTOOLSPACE             2003 ...
2006-02-13-12.30.37.181478 SYSTOOLSTMPSPACE         2001 ...
2006-02-13-12.30.37.181478 SYSTOOLSTMPSPACE         2001 ...
2006-02-13-12.30.37.181478 TEMPSPACE1               2001 ...
2006-02-13-12.30.37.181478 TEMPSPACE1               2001 ...
2006-02-13-12.30.37.181478 USERSPACE1               2001 ...
2006-02-13-12.30.37.181478 USERSPACE1               2001 ...
2006-02-13-12.30.37.181478 USERSPACE1               2002 ...
2006-02-13-12.30.37.181478 USERSPACE1               2002 ...
2006-02-13-12.30.37.181478 USERSPACE1               2003 ...
2006-02-13-12.30.37.181478 USERSPACE1               2003 ...

  22 record(s) selected.
```

Output from this query (continued).

```
... HI_TIMESTAMP              HI_VALUE HI_ALERT_STATE HI_ALERT_STATE_DETAIL ...
... -------------------------- -------- -------------- -------------------- ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
... 2006-02-13-12.16.25.911000       99              4 Alarm                ...
... 2006-02-13-12.06.26.168000       99              4 Alarm                ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
... 2006-02-13-12.16.25.911000       62              1 Normal               ...
... 2006-02-13-12.06.26.168000       62              1 Normal               ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
... 2006-02-13-12.16.25.911000      100              4 Alarm                ...
... 2006-02-13-12.06.26.168000      100              4 Alarm                ...
... 2006-02-13-12.16.25.911000        0              1 Normal               ...
... 2006-02-13-12.06.26.168000        0              1 Normal               ...
```

Output from this query (continued).

```
... HI_FORMULA              HI_ADDITIONAL_INFO
... --------------------... -------------------...---
... 0                       -
... 0                       -
... ((9376 / 9468) * 100)   The short term table space growth rate from
                              "02/13/2006 11:16:25.000911" to
                              "02/13/2006 12:16:25.000911" is "N/A" bytes
                              per second and the long term growth rate
                              from "02/12/2006 12:16:25.000911" to
                              "02/13/2006 12:16:25.000911" is "N/A" bytes
                              per second. Time to fullness is projected
                              to be "N/A" and "N/A" respectively. The
                              table space is defined with automatic
                              storage set to "YES" and automatic resize
                              enabled set to "YES".
... ((9376 / 9468) * 100)   The short term table space growth rate from
                              "02/13/2006 11:06:26.000168" to
                              "02/13/2006 12:06:26.000168" is "N/A" bytes
                              per second and the long term growth rate
                              from "02/12/2006 12:06:26.000168" to
                              "02/13/2006 12:06:26.000168" is "N/A" bytes
                              per second. Time to fullness is projected
                              to be "N/A" and "N/A" respectively. The
                              table space is defined with automatic
                              storage set to "YES" and automatic resize
                              enabled set to "YES".
... 0                       The table space is defined with automatic
                              storage set to "YES" and automatic resize
                              enabled set to "YES". The following are
                              the automatic resize settings: increase
                              size (bytes) "-1", increase size (percent)
                              "N/A", maximum size (bytes) "-1". The
                              current table space size (bytes) is
                              "38797312".
... 0                       The table space is defined with automatic
                              storage set to "YES" and automatic resize
                              enabled set to "YES". The following are
                              the automatic resize settings: increase
                              size (bytes) "-1", increase size (percent)
                              "N/A", maximum size (bytes) "-1". The
```

```
                                current table space size (bytes) is
                                "38797312".
... 0                           -
... 0                           -
... ((156 / 252) * 100)         The short term table space growth rate from
                                "02/13/2006 11:16:25.000911" to
                                "02/13/2006 12:16:25.000911" is "N/A"
                                bytes per second and the long term growth
                                rate from "02/12/2006 12:16:25.000911" to
                                "02/13/2006 12:16:25.000911" is "N/A" bytes
                                per second. Time to fullness is projected
                                to be "N/A" and "N/A" respectively. The
                                table space is defined with automatic
                                storage set to "YES" and automatic resize
                                enabled set to "YES".
... ((156 / 252) * 100)         The short term table space growth rate from
                                "02/13/2006 11:06:26.000168" to
                                "02/13/2006 12:06:26.000168" is "N/A"
                                bytes per second and the long term growth
                                rate from "02/12/2006 12:06:26.000168" to
                                "02/13/2006 12:06:26.000168" is "N/A" bytes
                                per second. Time to fullness is projected
                                to be "N/A" and "N/A" respectively. The
                                table space is defined with automatic
                                storage set to "YES" and automatic resize
                                enabled set to "YES".
... 0                           The table space is defined with automatic
                                storage set to "YES" and automatic resize
                                enabled set to "YES". The following are
                                the automatic resize settings: increase
                                size (bytes) "-1", increase size (percent)
                                "N/A", maximum size (bytes) "-1". The
                                current table space size (bytes) is
                                "1048576".
... 0                           The table space is defined with automatic
                                storage set to "YES" and automatic resize
                                enabled set to "YES". The following are
                                the automatic resize settings: increase
                                size (bytes) "-1", increase size (percent)
                                "N/A", maximum size (bytes) "-1". The
                                current table space size (bytes) is
                                "1048576".
... 0                           -
... 0                           -
... 0                           -
... 0                           -
... 0                           -
... 0                           -
... ((1504 / 1504) * 100)       The short term table space growth rate
                                from "02/13/2006 11:16:25.000911" to
                                "02/13/2006 12:16:25.000911" is "N/A"
                                bytes per second and the long term growth
                                rate from "02/12/2006 12:16:25.000911"
                                to "02/13/2006 12:16:25.000911" is "N/A"
                                bytes per second. Time to fullness is
                                projected to be "N/A" and "N/A"
                                respectively. The table space is defined
                                with automatic storage set to "YES" and
                                automatic resize enabled set to "YES".
... ((1504 / 1504) * 100)       The short term table space growth rate
                                from "02/13/2006 11:06:26.000168" to
                                "02/13/2006 12:06:26.000168" is "N/A"
                                bytes per second and the long term growth
                                rate from "02/12/2006 12:06:26.000168"
                                to "02/13/2006 12:06:26.000168" is "N/A"
                                bytes per second. Time to fullness is
                                projected to be "N/A" and "N/A"
```

```
                                 respectively. The table space is defined
                                 with automatic storage set to "YES" and
                                 automatic resize enabled set to "YES".
... 0                          The table space is defined with automatic
                                 storage set to "YES" and automatic
                                 resize enabled set to "YES". The
                                 following are the automatic resize
                                 settings: increase size (bytes) "-1",
                                 increase size (percent) "N/A", maximum
                                 size (bytes) "-1". The current table
                                 space size (bytes) is "6291456".
... 0                          The table space is defined with automatic
                                 storage set to "YES" and automatic
                                 resize enabled set to "YES". The
                                 following are the automatic resize
                                 settings: increase size (bytes) "-1",
                                 increase size (percent) "N/A", maximum
                                 size (bytes) "-1". The current table
                                 space size (bytes) is "6291456".
```

Information returned

Table 289. Information returned by the HEALTH_TBS_HI_HIS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLESPACE_NAME	VARCHAR(128)	tablespace_name - Table space name
HI_ID	BIGINT	A number that uniquely identifies the health indicator in the snapshot data stream.
HI_TIMESTAMP	TIMESTAMP	The date and time that the alert was generated.
HI_VALUE	SMALLINT	The value of the health indicator.
HI_ALERT_STATE	BIGINT	The severity of the alert.
HI_ALERT_STATE_DETAIL	VARCHAR(20)	The text description of the HI_ALERT_STATE column.
HI_FORMULA	VARCHAR(2048)	The formula used to calculate the health indicator.
HI_ADDITIONAL_INFO	VARCHAR(4096)	Additional information about the health indicator.

HEALTH_TBS_INFO

Returns table space information from a health snapshot of a database.

Important: This table function has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. For more information, see the "Health monitor has been deprecated" topic in the *What's New for DB2 Version 9.7* book.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the HEALTH_TBS_INFO table function.

Example

```
SELECT * FROM TABLE(HEALTH_TBS_INFO('',-1)) AS T
```

The following is an example of output from this query.

```
SNAPSHOT_TIMESTAMP         TABLESPACE_NAME      ...
-------------------------- ----------------...- ...
2006-02-13-12.30.35.027383 SYSCATSPACE          ...
2006-02-13-12.30.35.027383 SYSTOOLSPACE         ...
2006-02-13-12.30.35.027383 SYSTOOLSTMPSPACE     ...
2006-02-13-12.30.35.027383 TEMPSPACE1           ...
2006-02-13-12.30.35.027383 USERSPACE1           ...

  5 record(s) selected.
```

Output from this query (continued).

```
... ROLLED_UP_ALERT_STATE ROLLED_UP_ALERT_STATE_DETAIL
... --------------------- ----------------------------
...                     4 Alarm
...                     1 Normal
...                     1 Normal
...                     1 Normal
...                     4 Alarm
```

Information returned

Table 290. Information returned by the HEALTH_TBS_INFO table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name
ROLLED_UP_ALERT_STATE	BIGINT	The most severe alert state captured by this snapshot.
ROLLED_UP_ALERT_ STATE_DETAIL	VARCHAR(20)	The text description of the ROLLED_UP_ALERT_STATE column.

SNAP_GET_APPL table function – Retrieve appl logical data group snapshot information

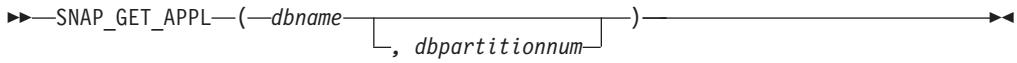
Note: This table function has been deprecated and replaced by the "SNAPAPPL administrative view and SNAP_GET_APPL_V95 table function - Retrieve appl logical data group snapshot information" on page 588.

The SNAP_GET_APPL table function returns information about applications from an application snapshot, in particular, the appl logical data group.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL_INFO, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_APPL table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 291 on page 1078 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_APPL table function takes a snapshot for the currently connected database and database partition number.

Authorization

- SYSMON authority
- EXECUTE privilege on the SNAP_GET_APPL table function.

Example

Retrieve details on rows read and written for each application for all active databases.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID, ROWS_READ, ROWS_WRITTEN
   FROM TABLE (SNAP_GET_APPL(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID       ROWS_READ      ROWS_WRITTEN
-------- --------...--- ---------...-- ------------...-
WSDB                679             0               0
WSDB                461             3               0
WSDB                460             4               0
TEST                680             4               0
TEST                455             6               0
TEST                454             0               0
TEST                453            50               0
```

Information returned

Table 291. Information returned by the SNAP_GET_APPL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
UOW_LOG_SPACE_USED	BIGINT	uow_log_space_used - Unit of work log space used
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
INACT_STMTHIST_SZ	BIGINT	stmt_history_list_size - Statement history list size

Table 291. Information returned by the SNAP_GET_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time

Table 291. Information returned by the SNAP_GET_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
TOTAL_SORTS	BIGINT	total_sorts - Total sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
COMMIT_SQL_STMTS	BIGINT	commit_sql_stmts - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	rollback_sql_stmts - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	dynamic_sql_stmts - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	static_sql_stmts - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	failed_sql_stmts - Failed statement operations
SELECT_SQL_STMTS	BIGINT	select_sql_stmts - Select SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements
UID_SQL_STMTS	BIGINT	uid_sql_stmts - UPDATE/INSERT/DELETE SQL statements executed
INT_AUTO_REBINDS	BIGINT	int_auto_rebinds - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
INT_COMMITS	BIGINT	int_commits - Internal commits
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	int_deadlock_rollbacks - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted

Table 291. Information returned by the SNAP_GET_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_SELECTED	BIGINT	rows_selected - Rows selected
BINDS_PRECOMPILES	BIGINT	binds_precompiles - Binds/precompiles attempted
OPEN_REM_CURS	BIGINT	open_rem_curs - Open remote cursors
OPEN_REM_CURS_BLK	BIGINT	open_rem_curs_blk - Open remote cursors with blocking
REJ_CURS_BLK	BIGINT	rej_curs_blk - Rejected block cursor requests
ACC_CURS_BLK	BIGINT	acc_curs_blk - Accepted block cursor requests
SQL_REQS_SINCE_COMMIT	BIGINT	sql_reqs_since_commit - SQL requests since last commit
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
OPEN_LOC_CURS	BIGINT	open_loc_curs - Open local cursors
OPEN_LOC_CURS_BLK	BIGINT	open_loc_curs_blk - Open local cursors with blocking
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	cat_cache_overflows - Catalog cache overflows
NUM_AGENTS	BIGINT	num_agents - Number of agents working on a statement
AGENTS_STOLEN	BIGINT	agents_stolen - Stolen agents
ASSOCIATED_AGENTS_TOP	BIGINT	associated_agents_top - Maximum number of associated agents
APPL_PRIORITY	BIGINT	appl_priority - Application agent priority
APPL_PRIORITY_TYPE	VARCHAR(16)	appl_priority_type - Application priority type. This interface returns a text identifier, based on defines in `sqlmon.h`, and is one of: • DYNAMIC_PRIORITY • FIXED_PRIORITY
PREFETCH_WAIT_TIME	BIGINT	prefetch_wait_time - Time waited for prefetch

Table 291. Information returned by the SNAP_GET_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
APPL_SECTION_LOOKUPS	BIGINT	appl_section_lookups - Section lookups
APPL_SECTION_INSERTS	BIGINT	appl_section_inserts - Section inserts
LOCKS_WAITING	BIGINT	locks_waiting - Current agents waiting on locks
TOTAL_HASH_JOINS	BIGINT	total_hash_joins - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	hash_join_overflows - Hash join overflows
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	hash_join_small_overflows - Hash join small overflows
APPL_IDLE_TIME	BIGINT	appl_idle_time - Application idle time
UOW_LOCK_WAIT_TIME	BIGINT	uow_lock_wait_time - Total time unit of work waited on locks
UOW_COMP_STATUS	VARCHAR(14)	uow_comp_status - Unit of work completion status. This interface returns a text identifier, based on defines in `sqlmon.h`, and is one of: • APPL_END • UOWABEND • UOWCOMMIT • UOWDEADLOCK • UOWLOCKTIMEOUT • UOWROLLBACK • UOWUNKNOWN
AGENT_USR_CPU_TIME_S	BIGINT	agent_usr_cpu_time - User CPU time used by agent (in seconds)*
AGENT_USR_CPU_TIME_MS	BIGINT	agent_usr_cpu_time - User CPU time used by agent (fractional, in microseconds)*
AGENT_SYS_CPU_TIME_S	BIGINT	agent_sys_cpu_time - System CPU time used by agent (in seconds)*
AGENT_SYS_CPU_TIME_MS	BIGINT	agent_sys_cpu_time - System CPU time used by agent (fractional, in microseconds)*
APPL_CON_TIME	TIMESTAMP	appl_con_time - Connection request start timestamp
CONN_COMPLETE_TIME	TIMESTAMP	conn_complete_time - Connection request completion timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
UOW_START_TIME	TIMESTAMP	uow_start_time - Unit of work start timestamp

Table 291. Information returned by the SNAP_GET_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
UOW_STOP_TIME	TIMESTAMP	uow_stop_time - Unit of work stop timestamp
PREV_UOW_STOP_TIME	TIMESTAMP	prev_uow_stop_time - Previous unit of work completion timestamp
UOW_ELAPSED_TIME_S	BIGINT	uow_elapsed_time - Most recent unit of work elapsed time (in seconds)*
UOW_ELAPSED_TIME_MS	BIGINT	uow_elapsed_time - Most recent unit of work elapsed time (fractional, in microseconds)*
ELAPSED_EXEC_TIME_S	BIGINT	elapsed_exec_time - Statement execution elapsed time (in seconds)*
ELAPSED_EXEC_TIME_MS	BIGINT	elapsed_exec_time - Statement execution elapsed time (fractional, in microseconds)*
INBOUND_COMM_ADDRESS	VARCHAR(32)	inbound_comm_address - Inbound communication address
LOCK_TIMEOUT_VAL	BIGINT	lock_timeout_val - Lock timeout (seconds)
PRIV_WORKSPACE_NUM_ OVERFLOWS	BIGINT	priv_workspace_num_overflows - Private workspace overflows
PRIV_WORKSPACE_SECTION_ INSERTS	BIGINT	priv_workspace_section_inserts - Private workspace section inserts
PRIV_WORKSPACE_SECTION_ LOOKUPS	BIGINT	priv_workspace_section_lookups - Private workspace section lookups
PRIV_WORKSPACE_SIZE_ TOP	BIGINT	priv_workspace_size_top - Maximum private workspace size
SHR_WORKSPACE_NUM_ OVERFLOWS	BIGINT	shr_workspace_num_overflows - Shared workspace overflows
SHR_WORKSPACE_SECTION_ INSERTS	BIGINT	shr_workspace_section_inserts - Shared workspace section inserts
SHR_WORKSPACE_SECTION_ LOOKUPS	BIGINT	shr_workspace_section_lookups - Shared workspace section lookups
SHR_WORKSPACE_SIZE_ TOP	BIGINT	shr_workspace_size_top - Maximum shared workspace size
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.
CAT_CACHE_SIZE_TOP	BIGINT	cat_cache_size_top - Catalog cache high water mark

Table 291. Information returned by the SNAP_GET_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAP_GET_APPL_INFO table function – Retrieve appl_info logical data group snapshot information

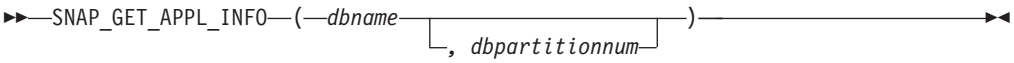
Note: This table function has been deprecated and replaced by the "SNAPAPPL_INFO administrative view and SNAP_GET_APPL_INFO_V95 table function - Retrieve appl_info logical data group snapshot information" on page 580.

The SNAP_GET_APPL_INFO table function returns information about applications from an application snapshot, in particular, the appl_info logical data group.

Used with the SNAP_GET_AGENT, SNAP_GET_AGENT_MEMORY_POOL, SNAP_GET_APPL, SNAP_GET_APPL_INFO, SNAP_GET_STMT and SNAP_GET_SUBSECTION table functions, the SNAP_GET_APPL_INFO table function provides information equivalent to the GET SNAPSHOT FOR ALL APPLICATIONS CLP command, but retrieves data from all database partitions.

Refer to Table 292 on page 1086 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all

active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_APPL_INFO table function takes a snapshot for the currently connected database and database partition number.

Authorization

- SYSMON authority
- EXECUTE privilege on the SNAP_GET_APPL_INFO table function.

Examples

Retrieve the status of all applications on the connected database partition.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AGENT_ID,
   SUBSTR(APPL_NAME,1,10) AS APPL_NAME, APPL_STATUS
   FROM TABLE(SNAP_GET_APPL_INFO(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AGENT_ID             APPL_NAME  APPL_STATUS
-------- -------------------- ---------- ----------------------
TOOLSDB                    14 db2bp.exe  CONNECTED
SAMPLE                     15 db2bp.exe  UOWEXEC
SAMPLE                      8 javaw.exe  CONNECTED
SAMPLE                      7 db2bp.exe  UOWWAIT

  4 record(s) selected.
```

The following shows what you obtain when you SELECT from the result of the table function.

```
SELECT SUBSTR(DB_NAME,1,8) AS DB_NAME, AUTHORITY_LVL
   FROM TABLE(SNAP_GET_APPL_INFO_V95(CAST(NULL AS VARCHAR(128)),-1)) AS T
```

The following is an example of output from this query.

```
DB_NAME  AUTHORITY_LVL
-------- -----------------------------------------------------....
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)
TESTDB   SYSADM(GROUP) + DBADM(USER) + CREATETAB(USER, GROUP) +
            BINDADD(USER, GROUP) + CONNECT(USER, GROUP) +
            CREATE_NOT_FENC(USER) + IMPLICIT_SCHEMA(USER, GROUP) +
            LOAD(USER) + CREATE_EXT_RT(USER) + QUIESCE_CONN(USER)

3 record(s) selected.
```

Information returned

Table 292. Information returned by the SNAP_GET_APPL_INFO table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
APPL_STATUS	VARCHAR(22)	appl_status - Application status. This interface returns a text identifier based on the defines in `sqlmon.h`, and is one of: • BACKUP • COMMIT_ACT • COMP • CONNECTED • CONNECTPEND • CREATE_DB • DECOUPLED • DISCONNECTPEND • INTR • IOERROR_WAIT • LOAD • LOCKWAIT • QUIESCE_TABLESPACE • RECOMP • REMOTE_RQST • RESTART • RESTORE • ROLLBACK_ACT • ROLLBACK_TO_SAVEPOINT • TEND • THABRT • THCOMT • TPREP • UNLOAD • UOWEXEC • UOWWAIT • WAITFOR_REMOTE
CODEPAGE_ID	BIGINT	codepage_id - ID of code page used by application
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
COORD_NODE_NUM	SMALLINT	coord_node - Coordinating node

Table 292. Information returned by the SNAP_GET_APPL_INFO table function (continued)

Column name	Data type	Description or corresponding monitor element
AUTHORITY_LVL	VARCHAR(512)	authority_lvl - User authorization level. This interface returns a text identifier based on the database authorities defined in `sql.h` and their source, and has the following format: authority(source, ...) + authority(source, ...) + ... The source of an authority can be multiple: either from a USER, a GROUP, or a USER and a GROUP. Possible values for "authority": • BINDADD • CONNECT • CREATE_EXT_RT • CREATE_NOT_FENC • CREATETAB • DBADM • IMPLICIT_SCHEMA • LOAD • LIBADM • QUIESCE_CONN • SECADM • SYSADM • SYSCTRL • SYSMAINT • SYSMON • SYSQUIESCE Possible values for "source": • USER – authority granted to the user or to a role granted to the user. • GROUP – authority granted to a group to which the user belongs or to a role granted to the group to which the user belongs.
CLIENT_PID	BIGINT	client_pid - Client process ID
COORD_AGENT_PID	BIGINT	coord_agent_pid - Coordinator agent
STATUS_CHANGE_TIME	TIMESTAMP	status_change_time - Application status change time

Table 292. Information returned by the SNAP_GET_APPL_INFO table function (continued)

Column name	Data type	Description or corresponding monitor element
CLIENT_PLATFORM	VARCHAR(12)	client_platform - Client operating platform. This interface returns a text identifier based on the defines in `sqlmon.h`, • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS • WINDOWS95

Table 292. Information returned by the SNAP_GET_APPL_INFO table function (continued)

Column name	Data type	Description or corresponding monitor element
CLIENT_PROTOCOL	VARCHAR(10)	client_protocol - Client communication protocol. This interface returns a text identifier based on the defines in `sqlmon.h`, • CPIC • LOCAL • NETBIOS • NPIPE • TCPIP (for DB2 UDB) • TCPIP4 • TCPIP6
TERRITORY_CODE	SMALLINT	territory_code - Database territory code
APPL_NAME	VARCHAR(256)	appl_name - Application name
APPL_ID	VARCHAR(128)	appl_id - Application ID
SEQUENCE_NO	VARCHAR(4)	sequence_no - Sequence number
PRIMARY_AUTH_ID	VARCHAR(128)	auth_id - Authorization ID
SESSION_AUTH_ID	VARCHAR(128)	session_auth_id - Session authorization ID
CLIENT_NNAME	VARCHAR(128)	The client_nname monitor element is deprecated. The value returned is not a valid value.
CLIENT_PRDID	VARCHAR(128)	client_prdid - Client product/version ID
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
CLIENT_DB_ALIAS	VARCHAR(128)	client_db_alias - Database alias used by application
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
EXECUTION_ID	VARCHAR(128)	execution_id - User login ID
CORR_TOKEN	VARCHAR(128)	corr_token - DRDA correlation token
TPMON_CLIENT_USERID	VARCHAR(256)	tpmon_client_userid - TP monitor client user ID
TPMON_CLIENT_WKSTN	VARCHAR(256)	tpmon_client_wkstn - TP monitor client workstation name
TPMON_CLIENT_APP	VARCHAR(256)	tpmon_client_app - TP monitor client application name
TPMON_ACC_STR	VARCHAR(200)	tpmon_acc_str - TP monitor client accounting string
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.

SNAP_GET_BP table function – Retrieve bufferpool logical group snapshot information

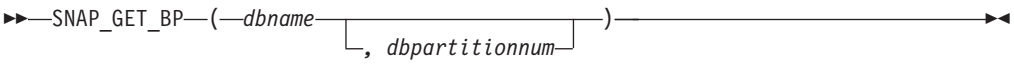
Note: This table function has been deprecated and replaced by the "SNAPBP administrative view and SNAP_GET_BP_V95 table function - Retrieve bufferpool logical group snapshot information" on page 596.

The SNAP_GET_BP table function returns information about buffer pools from a bufferpool snapshot, in particular, the bufferpool logical data group.

Used with the SNAP_GET_BP_PART table function, the SNAP_GET_BP table function provides the data equivalent to the GET SNAPSHOT FOR ALL BUFFERPOOLS CLP command.

Refer to Table 293 on page 1091 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a `NULL` value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_BP table function takes a snapshot for the currently connected database and database partition number.

Authorization

- SYSMON authority
- EXECUTE privilege on the SNAP_GET_BP table function.

Example

Retrieve total physical and logical reads for all bufferpools for all active databases for the currently connected database partition.

```
SELECT SUBSTR(T.DB_NAME,1,10) AS DB_NAME,
   SUBSTR(T.BP_NAME,1,20) AS BP_NAME,
   (T.POOL_DATA_L_READS+T.POOL_INDEX_L_READS) AS TOTAL_LOGICAL_READS,
   (T.POOL_DATA_P_READS+T.POOL_INDEX_P_READS) AS TOTAL_PHYSICAL_READS,
   T.DBPARTITIONNUM
   FROM TABLE(SNAP_GET_BP(CAST(NULL AS VARCHAR(128)), -1)) AS T
```

The following is an example of output from this query.

```
DB_NAME    BP_NAME          TOTAL_LOGICAL_READS  ...
---------- ------------...- -------------------- ...
SAMPLE     IBMDEFAULTBP                        0 ...
TOOLSDB    IBMDEFAULTBP                        0 ...
TOOLSDB    BP32K0000                           0 ...

  3 record(s) selected.
```

Output from this query (continued).

```
... TOTAL_PHYSICAL_READS DBPARTITIONNUM
... -------------------- --------------
...                    0              0
...                    0              0
...                    0              0
```

Information returned

Table 293. Information returned by the SNAP_GET_BP table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads

Table 293. Information returned by the SNAP_GET_BP table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time

Table 293. Information returned by the SNAP_GET_BP table function (continued)

Column name	Data type	Description or corresponding monitor element
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
FILES_CLOSED	BIGINT	files_closed - Database files closed
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
PAGES_FROM_BLOCK_IOS	BIGINT	pages_from_block_ios - Total number of pages read by block I/O
PAGES_FROM_VECTORED_IOS	BIGINT	pages_from_vectored_ios - Total pages read by vectored I/O
PHYSICAL_PAGE_MAPS	BIGINT	The physical_page_maps monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
VECTORED_IOS	BIGINT	vectored_ios - Number of vectored I/O requests
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAP_GET_CONTAINER

Note: This table function has been deprecated and replaced by the "SNAPCONTAINER administrative view and SNAP_GET_CONTAINER_V91 table function - Retrieve tablespace_container logical data group snapshot information" on page 605

The schema is SYSPROC.

The SNAP_GET_CONTAINER table function returns snapshot information from the tablespace_container logical data group.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition. If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 294. Information returned by the SNAP_GET_CONTAINER table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
CONTAINER_NAME	VARCHAR(256)	container_name - Container name
CONTAINER_ID	BIGINT	container_id - Container identification
CONTAINER_TYPE	SMALLINT	container_type - Container type
TOTAL_PAGES	BIGINT	container_total_pages - Total pages in container
USABLE_PAGES	BIGINT	container_usable_pages - Usable pages in container
ACCESSIBLE	SMALLINT	container_accessible - Accessibility of container
STRIPE_SET	BIGINT	container_stripe_set - Stripe set
DBPARTITIONNUM	SMALLINT	node_number - Node number

SNAP_GET_DB

Note: This table function has been deprecated and replaced by the "SNAP_GET_DB_V91 table function - Retrieve snapshot information from the dbase logical group" on page 1105

The schema is SYSPROC.

The SNAP_GET_DB table function returns snapshot information from the database.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 295. Information returned by the SNAP_GET_DB table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	**db_name** - Database name
DB_PATH	VARCHAR(1024)	**db_path** - Database path
INPUT_DB_ALIAS	VARCHAR(128)	**input_db_alias** - Input database alias
DB_STATUS	BIGINT	**db_status** - Status of database
CATALOG_PARTITION	SMALLINT	**catalog_node** - Catalog node number
CATALOG_PARTITION_NAME	VARCHAR(128)	**catalog_node_name** - Catalog node network name
SERVER_PLATFORM	INTEGER	**server_platform** - Server operating system
DB_LOCATION	INTEGER	**db_location** - Database location
DB_CONN_TIME	TIMESTAMP	**db_conn_time** - Database activation timestamp

Table 295. Information returned by the SNAP_GET_DB table function (continued)

Column name	Data type	Description or corresponding monitor element
LAST_RESET	TIMESTAMP	**last_reset** - Last reset timestamp
LAST_BACKUP	TIMESTAMP	**last_backup** - Last backup timestamp
CONNECTIONS_TOP	BIGINT	**connections_top** - Maximum number of concurrent connections
TOTAL_CONS	BIGINT	**total_cons** - Connects since database activation
TOTAL_SEC_CONS	BIGINT	**total_sec_cons** - Secondary connections
APPLS_CUR_CONS	BIGINT	**appls_cur_cons** - Applications connected currently
APPLS_IN_DB2	BIGINT	**appls_in_db2** - Applications executing in the database currently
NUM_ASSOC_AGENTS	BIGINT	**num_assoc_agents** - Number of associated agents
AGENTS_TOP	BIGINT	**agents_top** - Number of agents created
COORD_AGENTS_TOP	BIGINT	**coord_agents_top** - Maximum number of coordinating agents
LOCKS_HELD	BIGINT	**locks_held** - Locks held
LOCK_WAITS	BIGINT	**lock_waits** - Lock waits
LOCK_WAIT_TIME	BIGINT	**lock_wait_time** - Time waited on locks
LOCK_LIST_IN_USE	BIGINT	**lock_list_in_use** - Total lock list memory in use
DEADLOCKS	BIGINT	**deadlocks** - Deadlocks detected
LOCK_ESCALS	BIGINT	**lock_escals** - Number of lock escalations
X_LOCK_ESCALS	BIGINT	**x_lock_escals** - Exclusive lock escalations
LOCKS_WAITING	BIGINT	**locks_waiting** - Current agents waiting on locks
LOCK_TIMEOUTS	BIGINT	**lock_timeouts** - Number of lock timeouts
NUM_INDOUBT_TRANS	BIGINT	**num_indoubt_trans** - Number of indoubt transactions
SORT_HEAP_ALLOCATED	BIGINT	**sort_heap_allocated** - Total sort heap allocated
SORT_SHRHEAP_ALLOCATED	BIGINT	**sort_shrheap_allocated** - Sort share heap currently allocated
SORT_SHRHEAP_TOP	BIGINT	**sort_shrheap_top** - Sort share heap high water mark
TOTAL_SORTS	BIGINT	**total_sorts** - Total sorts
TOTAL_SORT_TIME	BIGINT	**total_sort_time** - Total sort time
SORT_OVERFLOWS	BIGINT	**sort_overflows** - Sort overflows

Table 295. Information returned by the SNAP_GET_DB table function (continued)

Column name	Data type	Description or corresponding monitor element
ACTIVE_SORTS	BIGINT	**active_sorts** - Active sorts
POOL_DATA_L_READS	BIGINT	**pool_data_l_reads** - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	**pool_data_p_reads** - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	**pool_temp_data_l_reads** - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	**pool_temp_data_p_reads** - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	**pool_async_data_reads** - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	**pool_data_writes** - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	**pool_async_data_writes** - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	**pool_index_l_reads** - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	**pool_index_p_reads** - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	**pool_temp_index_l_reads** - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	**pool_temp_index_p_reads** - Buffer pool temporary index physical reads
POOL_INDEX_WRITES	BIGINT	**pool_index_writes** - Buffer pool index writes
POOL_ASYNC_INDEX_READS	BIGINT	**pool_async_index_reads** - Buffer pool asynchronous index reads
POOL_ASYNC_INDEX_WRITES	BIGINT	**pool_async_index_writes** - Buffer pool asynchronous index writes
POOL_READ_TIME	BIGINT	**pool_read_time** - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	**pool_write_time** - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	**pool_async_read_time** - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	**pool_async_write_time** - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	**pool_async_data_read_reqs** - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	**pool_async_index_read_reqs** - Buffer pool asynchronous index read requests
POOL_NO_VICTIM_BUFFER	BIGINT	**pool_no_victim_buffer** - Buffer pool no victim buffers

Table 295. Information returned by the SNAP_GET_DB table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_LSN_GAP_CLNS	BIGINT	**pool_lsn_gap_clns** - Buffer pool log space cleaners triggered
POOL_DRTY_PG_STEAL_CLNS	BIGINT	**pool_drty_pg_steal_clns** - Buffer pool victim page cleaners triggered
POOL_DRTY_PG_THRSH_CLNS	BIGINT	**pool_drty_pg_thrsh_clns** - Buffer pool threshold cleaners triggered
PREFETCH_WAIT_TIME	BIGINT	**prefetch_wait_time** - Time waited for prefetch
UNREAD_PREFETCH_PAGES	BIGINT	**unread_prefetch_pages** - Unread prefetch pages
DIRECT_READS	BIGINT	**direct_reads** - Direct reads from database
DIRECT_WRITES	BIGINT	**direct_writes** - Direct writes to database
DIRECT_READ_REQS	BIGINT	**direct_read_reqs** - Direct read requests
DIRECT_WRITE_REQS	BIGINT	**direct_write_reqs** - Direct write requests
DIRECT_READ_TIME	BIGINT	**direct_read_time** - Direct read time
DIRECT_WRITE_TIME	BIGINT	**direct_write_time** - Direct write time
FILES_CLOSED	BIGINT	**files_closed** - Database files closed
POOL_DATA_TO_ESTORE	BIGINT	The **pool_data_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_TO_ESTORE	BIGINT	The **pool_index_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_FROM_ESTORE	BIGINT	The **pool_index_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_DATA_FROM_ESTORE	BIGINT	The **pool_data_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
ELAPSED_EXEC_TIME_S	BIGINT	**elapsed_exec_time** - Statement execution elapsed time (in seconds)*
ELAPSED_EXEC_TIME_MS	BIGINT	**elapsed_exec_time** - Statement execution elapsed time (fractional, in microseconds)*
COMMIT_SQL_STMTS	BIGINT	**commit_sql_stmts** - Commit statements attempted

Table 295. Information returned by the SNAP_GET_DB table function (continued)

Column name	Data type	Description or corresponding monitor element
ROLLBACK_SQL_STMTS	BIGINT	**rollback_sql_stmts** - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	**dynamic_sql_stmts** - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	**static_sql_stmts** - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	**failed_sql_stmts** - Failed statement operations
SELECT_SQL_STMTS	BIGINT	**select_sql_stmts** - Select SQL statements executed
UID_SQL_STMTS	BIGINT	**uid_sql_stmts** - UPDATE/INSERT/DELETE SQL statements executed
DDL_SQL_STMTS	BIGINT	**ddl_sql_stmts** - Data definition language (DDL) SQL statements
INT_AUTO_REBINDS	BIGINT	**int_auto_rebinds** - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	**int_rows_deleted** - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	**int_rows_inserted** - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	**int_rows_updated** - Internal rows updated
INT_COMMITS	BIGINT	**int_commits** - Internal commits
INT_ROLLBACKS	BIGINT	**int_rollbacks** - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	**int_deadlock_rollbacks** - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	**rows_deleted** - Rows deleted
ROWS_INSERTED	BIGINT	**rows_inserted** - Rows inserted
ROWS_UPDATED	BIGINT	**rows_updated** - Rows updated
ROWS_SELECTED	BIGINT	**rows_selected** - Rows selected
ROWS_READ	BIGINT	**rows_read** - Rows read
BINDS_PRECOMPILES	BIGINT	**binds_precompiles** - Binds/precompiles attempted
TOTAL_LOG_AVAILABLE	BIGINT	**total_log_available** - Total log available
TOTAL_LOG_USED	BIGINT	**total_log_used** - Total log space used
SEC_LOG_USED_TOP	BIGINT	**sec_log_used_top** - Maximum secondary log space used
TOT_LOG_USED_TOP	BIGINT	**tot_log_used_top** - Maximum total log space used
SEC_LOGS_ALLOCATED	BIGINT	**sec_logs_allocated** - Secondary logs allocated currently

Table 295. Information returned by the SNAP_GET_DB table function (continued)

Column name	Data type	Description or corresponding monitor element
LOG_READS	BIGINT	**log_reads** - Number of log pages read
LOG_READ_TIME_S	BIGINT	**log_read_time** - Log read time (in seconds)†
LOG_READ_TIME_NS	BIGINT	**log_read_time** - Log read time (fractional, in nanoseconds)†
LOG_WRITES	BIGINT	**log_writes** - Number of log pages written
LOG_WRITE_TIME_S	BIGINT	**log_write_time** - Log write time (in seconds)†
LOG_WRITE_TIME_NS	BIGINT	**log_write_time** - Log write time(fractional, in nanoseconds)†
NUM_LOG_WRITE_IO	BIGINT	**num_log_write_io** - Number of log writes
NUM_LOG_READ_IO	BIGINT	**num_log_read_io** - Number of log reads
NUM_LOG_PART_PAGE_IO	BIGINT	**num_log_part_page_io** - Number of partial log page writes
NUM_LOG_BUFFER_FULL	BIGINT	**num_log_buffer_full** - Number of full log buffers
NUM_LOG_DATA_FOUND_ IN_BUFFER	BIGINT	**num_log_data_found_in_buffer** - Number of log data found in buffer
APPL_ID_OLDEST_XACT	BIGINT	**appl_id_oldest_xact** - Application with oldest transaction
LOG_TO_REDO_FOR_RECOVERY	BIGINT	**log_to_redo_for_recovery** - Amount of log to be redone for recovery
LOG_HELD_BY_DIRTY_PAGES	BIGINT	**log_held_by_dirty_pages** - Amount of log space accounted for by dirty pages
PKG_CACHE_LOOKUPS	BIGINT	**pkg_cache_lookups** - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	**pkg_cache_inserts** - Package cache inserts
PKG_CACHE_NUM_ OVERFLOWS	BIGINT	**pkg_cache_num_overflows** - Package cache overflows
PKG_CACHE_SIZE_TOP	BIGINT	**pkg_cache_size_top** - Package cache high water mark
APPL_SECTION_LOOKUPS	BIGINT	**appl_section_lookups** - Section lookups
APPL_SECTION_INSERTS	BIGINT	**appl_section_inserts** - Section inserts
CAT_CACHE_LOOKUPS	BIGINT	**cat_cache_lookups** - Catalog cache lookups

Table 295. Information returned by the SNAP_GET_DB table function (continued)

Column name	Data type	Description or corresponding monitor element
CAT_CACHE_INSERTS	BIGINT	**cat_cache_inserts** - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	**cat_cache_overflows** - Catalog cache overflows
CAT_CACHE_SIZE_TOP	BIGINT	**cat_cache_size_top** - Catalog cache high water mark
PRIV_WORKSPACE_SIZE_TOP	BIGINT	**priv_workspace_size_top** - Maximum private workspace size
PRIV_WORKSPACE_NUM_ OVERFLOWS	BIGINT	**priv_workspace_num_overflows** - Private workspace overflows
PRIV_WORKSPACE_SECTION_ INSERTS	BIGINT	**priv_workspace_section_inserts** - Private workspace section inserts
PRIV_WORKSPACE_SECTION_ LOOKUPS	BIGINT	**priv_workspace_section_lookups** - Private workspace section lookups
SHR_WORKSPACE_SIZE_TOP	BIGINT	**shr_workspace_size_top** - Maximum shared workspace size
SHR_WORKSPACE_NUM_ OVERFLOWS	BIGINT	**shr_workspace_num_overflows** - Shared workspace overflows
SHR_WORKSPACE_SECTION_ INSERTS	BIGINT	**shr_workspace_section_inserts** - Shared workspace section inserts
SHR_WORKSPACE_SECTION_ LOOKUPS	BIGINT	**shr_workspace_section_lookups** - Shared workspace section lookups
TOTAL_HASH_JOINS	BIGINT	**total_hash_joins** - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	**total_hash_loops** - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	**hash_join_overflows** - Hash join overflows
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	**hash_join_small_overflows** - Hash join small overflows
NUM_DB_STORAGE_PATHS	BIGINT	**num_db_storage_paths** - Number of automatic storage paths
DBPARTITIONNUM	SMALLINT	**node_number** - Node number

Table 295. Information returned by the SNAP_GET_DB table function (continued)

Column name	Data type	Description or corresponding monitor element
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000. †To calculate the total elapsed time for this monitor element, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000,000 + *monitor-element-name*_MS) ÷ 1,000,000,000. For example, (LOG_READ_TIME_S × 1,000,000,000 + LOG_READ_TIME_MS) ÷ 1,000,000,000.		

SNAP_GET_DBM table function – Retrieve the dbm logical grouping snapshot information

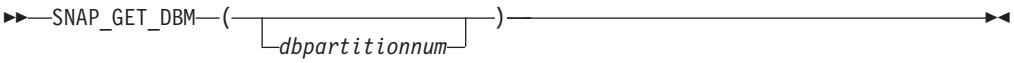
Note: This table function has been deprecated and replaced by the "SNAPDBM administrative view and SNAP_GET_DBM_V95 table function - Retrieve the dbm logical grouping snapshot information" on page 624.

The SNAP_GET_DBM table function returns the snapshot monitor DB2 database manager (dbm) logical grouping information.

Used with the SNAP_GET_DBM_MEMORY_POOL, SNAP_GET_FCM, SNAP_GET_FCM_PART and SNAP_GET_SWITCHES table functions, the SNAP_GET_DBM table function provides the data equivalent to the GET SNAPSHOT FOR DBM command.

Refer to Table 296 on page 1103 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameter

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If this input option is not used, data will be returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If *dbpartitionnum* is set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been

created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DBM table function calls the snapshot from memory.

Authorization

- SYSMON authority
- EXECUTE privilege on the SNAP_GET_DBM table function.

Example

Retrieve the start time and current status of database partition number 2.

```
SELECT DB2START_TIME, DB2_STATUS FROM TABLE(SNAP_GET_DBM(2)) AS T
```

The following is an example of output from this query.

```
DB2START_TIME              DB2_STATUS
-------------------------- ------------
2006-01-06-14.59.59.062798 ACTIVE
```

Information returned

Table 296. Information returned by the SNAP_GET_DBM table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
SORT_HEAP_ALLOCATED	BIGINT	sort_heap_allocated - Total sort heap allocated
POST_THRESHOLD_SORTS	BIGINT	post_threshold_sorts - Post threshold sorts
PIPED_SORTS_REQUESTED	BIGINT	piped_sorts_requested - Piped sorts requested
PIPED_SORTS_ACCEPTED	BIGINT	piped_sorts_accepted - Piped sorts accepted
REM_CONS_IN	BIGINT	rem_cons_in - Remote connections to database manager
REM_CONS_IN_EXEC	BIGINT	rem_cons_in_exec - Remote Connections Executing in the Database Manager monitor element
LOCAL_CONS	BIGINT	local_cons - Local connections
LOCAL_CONS_IN_EXEC	BIGINT	local_cons_in_exec - Local Connections Executing in the Database Manager monitor element
CON_LOCAL_DBASES	BIGINT	con_local_dbases - Local databases with current connects
AGENTS_REGISTERED	BIGINT	agents_registered - Agents registered
AGENTS_WAITING_ON_TOKEN	BIGINT	agents_waiting_on_token - Agents waiting for a token

Table 296. Information returned by the SNAP_GET_DBM table function (continued)

Column name	Data type	Description or corresponding monitor element
DB2_STATUS	VARCHAR(12)	db2_status - Status of DB2 instance. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ACTIVE • QUIESCE_PEND • QUIESCED
AGENTS_REGISTERED_TOP	BIGINT	agents_registered_top - Maximum number of agents registered
AGENTS_WAITING_TOP	BIGINT	agents_waiting_top - Maximum number of agents waiting
COMM_PRIVATE_MEM	BIGINT	comm_private_mem - Committed private memory
IDLE_AGENTS	BIGINT	idle_agents - Number of idle agents
AGENTS_FROM_POOL	BIGINT	agents_from_pool - Agents assigned from pool
AGENTS_CREATED_EMPTY_POOL	BIGINT	agents_created_empty_pool - Agents created due to empty agent pool
COORD_AGENTS_TOP	BIGINT	coord_agents_top - Maximum number of coordinating agents
MAX_AGENT_OVERFLOWS	BIGINT	max_agent_overflows - Maximum agent overflows
AGENTS_STOLEN	BIGINT	agents_stolen - Stolen agents
GW_TOTAL_CONS	BIGINT	gw_total_cons - Total number of attempted connections for DB2 Connect
GW_CUR_CONS	BIGINT	gw_cur_cons - Current number of connections for DB2 Connect
GW_CONS_WAIT_HOST	BIGINT	gw_cons_wait_host - Number of connections waiting for the host to reply
GW_CONS_WAIT_CLIENT	BIGINT	gw_cons_wait_client - Number of connections waiting for the client to send request
POST_THRESHOLD_HASH_JOINS	BIGINT	post_threshold_hash_joins - Hash join threshold
NUM_GW_CONN_SWITCHES	BIGINT	num_gw_conn_switches - Connection switches
DB2START_TIME	TIMESTAMP	db2start_time - Start database manager timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
NUM_NODES_IN_DB2_INSTANCE	INTEGER	num_nodes_in_db2_instance - Number of nodes in database partition

Table 296. Information returned by the SNAP_GET_DBM table function (continued)

Column name	Data type	Description or corresponding monitor element
PRODUCT_NAME	VARCHAR(32)	product_name - Product name
SERVICE_LEVEL	VARCHAR(18)	service_level - Service level
SORT_HEAP_TOP	BIGINT	sort_heap_top - Sort private heap high water mark
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAP_GET_DB_V91 table function - Retrieve snapshot information from the dbase logical group

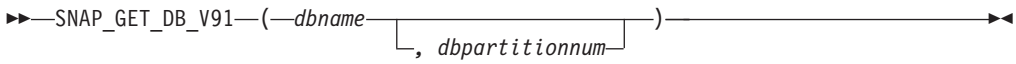
Note: This table function has been deprecated and replaced by the "SNAPDB administrative view and SNAP_GET_DB_V95 table function - Retrieve snapshot information from the dbase logical group" on page 749.

The SNAP_GET_DB_V91 table function returns snapshot information from the database (dbase) logical group.

Used in conjunction with the SNAP_GET_DB_MEMORY_POOL, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DB_V91 table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 297 on page 1107 for a complete list of information that is returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all

active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DB_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorities or privilege is required:

- EXECUTE privilege on the SNAP_GET_DB_V91 table function.
- DATAACCESS authority
- SYSMON authority
- SYSMAINT authority
- SYSCTRL authority
- SYSADM authority

Examples

Example 1: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions of the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME FROM TABLE(SNAP_GET_DB_V91('', -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME     DB_STATUS    SERVER_PLATFORM ...
-------...- ------------ --------------- ...
SAMPLE      ACTIVE       AIX64           ...

  1 record(s) selected.
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.09.22.013196
```

Example 2: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions for all active databases in the same instance that contains the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME
   FROM TABLE(SNAP_GET_DB_V91(CAST (NULL AS VARCHAR(128)), -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME      DB_STATUS    SERVER_PLATFORM ...
--------...- ------------ --------------- ...
TOOLSDB      ACTIVE       AIX64           ...
SAMPLE       ACTIVE       AIX64           ...
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.26.54.396335
... LOCAL        2005-07-24-22.09.22.013196
```

SNAP_GET_DB_V91 table function metadata

Table 297. Information returned by the SNAP_GET_DB_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
DB_STATUS	VARCHAR(12)	db_status - Status of database. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ACTIVE • QUIESCE_PEND • QUIESCED • ROLLFWD
CATALOG_PARTITION	SMALLINT	catalog_node - Catalog node number
CATALOG_PARTITION_NAME	VARCHAR(128)	catalog_node_name - Catalog node network name

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
SERVER_PLATFORM	VARCHAR(12)	server_platform - Server operating system. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS • WINDOWS95
DB_LOCATION	VARCHAR(12)	db_location - Database location. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • LOCAL • REMOTE
DB_CONN_TIME	TIMESTAMP	db_conn_time - Database activation timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
LAST_BACKUP	TIMESTAMP	last_backup - Last backup timestamp
CONNECTIONS_TOP	BIGINT	connections_top - Maximum number of concurrent connections
TOTAL_CONS	BIGINT	total_cons - Connects since database activation
TOTAL_SEC_CONS	BIGINT	total_sec_cons - Secondary connections
APPLS_CUR_CONS	BIGINT	appls_cur_cons - Applications connected currently
APPLS_IN_DB2	BIGINT	appls_in_db2 - Applications executing in the database currently
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
AGENTS_TOP	BIGINT	agents_top - Number of agents created
COORD_AGENTS_TOP	BIGINT	coord_agents_top - Maximum number of coordinating agents
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_LIST_IN_USE	BIGINT	lock_list_in_use - Total lock list memory in use
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
LOCKS_WAITING	BIGINT	locks_waiting - Current agents waiting on locks
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
NUM_INDOUBT_TRANS	BIGINT	num_indoubt_trans - Number of indoubt transactions
SORT_HEAP_ALLOCATED	BIGINT	sort_heap_allocated - Total sort heap allocated
SORT_SHRHEAP_ALLOCATED	BIGINT	sort_shrheap_allocated - Sort share heap currently allocated
SORT_SHRHEAP_TOP	BIGINT	sort_shrheap_top - Sort share heap high water mark
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
TOTAL_SORTS	BIGINT	total_sorts - Total sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
ACTIVE_SORTS	BIGINT	active_sorts - Active sorts
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
POOL_LSN_GAP_CLNS	BIGINT	pool_lsn_gap_clns - Buffer pool log space cleaners triggered
POOL_DRTY_PG_STEAL_CLNS	BIGINT	pool_drty_pg_steal_clns - Buffer pool victim page cleaners triggered
POOL_DRTY_PG_THRSH_CLNS	BIGINT	pool_drty_pg_thrsh_clns - Buffer pool threshold cleaners triggered
PREFETCH_WAIT_TIME	BIGINT	prefetch_wait_time - Time waited for prefetch
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
FILES_CLOSED	BIGINT	files_closed - Database files closed
ELAPSED_EXEC_TIME_S	BIGINT	elapsed_exec_time - Statement execution elapsed time (in seconds)*

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
ELAPSED_EXEC_TIME_MS	BIGINT	elapsed_exec_time - Statement execution elapsed time (fractional, in microseconds)*
COMMIT_SQL_STMTS	BIGINT	commit_sql_stmts - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	rollback_sql_stmts - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	dynamic_sql_stmts - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	static_sql_stmts - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	failed_sql_stmts - Failed statement operations
SELECT_SQL_STMTS	BIGINT	select_sql_stmts - Select SQL statements executed
UID_SQL_STMTS	BIGINT	uid_sql_stmts - UPDATE/INSERT/DELETE SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements
INT_AUTO_REBINDS	BIGINT	int_auto_rebinds - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
INT_COMMITS	BIGINT	int_commits - Internal commits
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	int_deadlock_rollbacks - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_SELECTED	BIGINT	rows_selected - Rows selected
ROWS_READ	BIGINT	rows_read - Rows read
BINDS_PRECOMPILES	BIGINT	binds_precompiles - Binds/precompiles attempted
TOTAL_LOG_AVAILABLE	BIGINT	total_log_available - Total log available
TOTAL_LOG_USED	BIGINT	total_log_used - Total log space used
SEC_LOG_USED_TOP	BIGINT	sec_log_used_top - Maximum secondary log space used

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
TOT_LOG_USED_TOP	BIGINT	tot_log_used_top - Maximum total log space used
SEC_LOGS_ALLOCATED	BIGINT	sec_logs_allocated - Secondary logs allocated currently
LOG_READS	BIGINT	log_reads - Number of log pages read
LOG_READ_TIME_S	BIGINT	log_read_time - Log read time (in seconds)†
LOG_READ_TIME_NS	BIGINT	log_read_time - Log read time (fractional, in nanoseconds)†
LOG_WRITES	BIGINT	log_writes - Number of log pages written
LOG_WRITE_TIME_S	BIGINT	log_write_time - Log write time (in seconds)†
LOG_WRITE_TIME_NS	BIGINT	log_write_time - Log write time (fractional, in nanoseconds)†
NUM_LOG_WRITE_IO	BIGINT	num_log_write_io - Number of log writes
NUM_LOG_READ_IO	BIGINT	num_log_read_io - Number of log reads
NUM_LOG_PART_PAGE_IO	BIGINT	num_log_part_page_io - Number of partial log page writes
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers
NUM_LOG_DATA_FOUND_ IN_BUFFER	BIGINT	num_log_data_found_in_buffer - Number of log data found in buffer
APPL_ID_OLDEST_XACT	BIGINT	appl_id_oldest_xact - Application with oldest transaction
LOG_TO_REDO_FOR_ RECOVERY	BIGINT	log_to_redo_for_recovery - Amount of log to be redone for recovery
LOG_HELD_BY_DIRTY_PAGES	BIGINT	log_held_by_dirty_pages - Amount of log space accounted for by dirty pages
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
PKG_CACHE_NUM_ OVERFLOWS	BIGINT	pkg_cache_num_overflows - Package cache overflows
PKG_CACHE_SIZE_TOP	BIGINT	pkg_cache_size_top - Package cache high water mark
APPL_SECTION_LOOKUPS	BIGINT	appl_section_lookups - Section lookups

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
APPL_SECTION_INSERTS	BIGINT	appl_section_inserts - Section inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	cat_cache_overflows - Catalog cache overflows
CAT_CACHE_SIZE_TOP	BIGINT	cat_cache_size_top - Catalog cache high water mark
PRIV_WORKSPACE_SIZE_TOP	BIGINT	priv_workspace_size_top - Maximum private workspace size
PRIV_WORKSPACE_NUM_ OVERFLOWS	BIGINT	priv_workspace_num_overflows - Private workspace overflows
PRIV_WORKSPACE_SECTION_ INSERTS	BIGINT	priv_workspace_section_inserts - Private workspace section inserts
PRIV_WORKSPACE_SECTION_ LOOKUPS	BIGINT	priv_workspace_section_lookups - Private workspace section lookups
SHR_WORKSPACE_SIZE_TOP	BIGINT	shr_workspace_size_top - Maximum shared workspace size
SHR_WORKSPACE_NUM_ OVERFLOWS	BIGINT	shr_workspace_num_overflows - Shared workspace overflows
SHR_WORKSPACE_SECTION_ INSERTS	BIGINT	shr_workspace_section_inserts - Shared workspace section inserts
SHR_WORKSPACE_SECTION_ LOOKUPS	BIGINT	shr_workspace_section_lookups - Shared workspace section lookups
TOTAL_HASH_JOINS	BIGINT	total_hash_joins - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	hash_join_overflows - Hash join overflows
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	hash_join_small_overflows - Hash join small overflows
POST_SHRTHRESHOLD_ HASH_JOINS	BIGINT	post_shrthreshold_hash_joins - Post threshold hash joins
ACTIVE_HASH_JOINS	BIGINT	active_hash_joins - Active hash joins
NUM_DB_STORAGE_PATHS	BIGINT	num_db_storage_paths - Number of automatic storage paths

Table 297. Information returned by the SNAP_GET_DB_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
SMALLEST_LOG_AVAIL_NODE	INTEGER	smallest_log_avail_node - Node with least available log space

* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.

†To calculate the total elapsed time for this monitor element, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000,000 + *monitor-element-name*_MS) ÷ 1,000,000,000. For example, (LOG_READ_TIME_S × 1,000,000,000 + LOG_READ_TIME_MS) ÷ 1,000,000,000.

SNAPDB administrative view and SNAP_GET_DB_V95 table function - Retrieve snapshot information from the dbase logical group

Note: The SNAP_GET_DB_V95 table function has been deprecated and replaced by the SNAP_GET_DB_V97 table function - Retrieve snapshot information from the dbase logical group..

The "SNAPDB administrative view" on page 749 and the "SNAP_GET_DB_V95 table function" on page 750 return snapshot information from the database (dbase) logical group.

SNAPDB administrative view

This administrative view allows you to retrieve snapshot information from the dbase logical group for the currently connected database.

Used in conjunction with the SNAPDB_MEMORY_POOL, SNAPDETAILLOG, SNAPHADR and SNAPSTORAGE_PATHS administrative views, the SNAPDB administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE on database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 206 on page 753 for a complete list of information that is returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPDB administrative view
- CONTROL privilege on the SNAPDB administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_DB_V95 table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Retrieve the status, platform, location, and connect time for all database partitions of the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDB ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
DB_NAME     DB_STATUS    SERVER_PLATFORM DB_LOCATION  ...
-------...- ------------ --------------- ------------ ...
TEST        ACTIVE       AIX64           LOCAL        ...
TEST        ACTIVE       AIX64           LOCAL        ...
TEST        ACTIVE       AIX64           LOCAL        ...

3 record(s) selected.
```

Output from this query (continued).

```
... DB_CONN_TIME               DBPARTITIONNUM
... -------------------------- --------------
... 2006-01-08-16.48.30.665477              0
... 2006-01-08-16.48.34.005328              1
... 2006-01-08-16.48.34.007937              2
```

This routine can be used by calling the following on the command line:

```
SELECT TOTAL_OLAP_FUNCS, OLAP_FUNC_OVERFLOWS, ACTIVE_OLAP_FUNCS
   FROM SYSIBMADM.SNAPDB

TOTAL_OLAP_FUNCS     OLAP_FUNC_OVERFLOWS  ACTIVE_OLAP_FUNCS
-------------------- -------------------- -----------------
                   7                    2                 1

  1 record(s) selected.
```

After running a workload, a user can use the following query:

```
SELECT STATS_CACHE_SIZE, STATS_FABRICATIONS, SYNC_RUNSTATS,
   ASYNC_RUNSTATS, STATS_FABRICATE_TIME, SYNC_RUNSTATS_TIME
   FROM SYSIBMADM.SNAPDB

STATS_CACHE_SIZE STATS_FABRICATIONS SYNC_RUNSTATS ASYNC_RUNSTATS ...
---------------- ------------------ ------------- -------------- ...
             128                  2             1              0 ...


... STATS_FABRICATE_TIME SYNC_RUNSTATS_TIME
```

```
... ------------------- -----------------
...                  10               100

1 record(s) selected.
```

SNAP_GET_DB_V95 table function

The SNAP_GET_DB_V95 table function returns the same information as the SNAPDB administrative view.

Used in conjunction with the SNAP_GET_DB_MEMORY_POOL, SNAP_GET_DETAILLOG_V91, SNAP_GET_HADR and SNAP_GET_STORAGE_PATHS table functions, the SNAP_GET_DB_V95 table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Refer to Table 206 on page 753 for a complete list of information that is returned.

Syntax

The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify an empty string to take the snapshot from the currently connected database. Specify a NULL value to take the snapshot from all databases within the same instance as the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DB_V95 table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_DB_V95 table function

- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Examples

Example 1: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions of the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME FROM TABLE(SNAP_GET_DB_V95('', -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME     DB_STATUS    SERVER_PLATFORM ...
-------...- ------------ --------------- ...
SAMPLE      ACTIVE       AIX64           ...

  1 record(s) selected.
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.09.22.013196
```

Example 2: Retrieve the status, platform, location, and connect time as an aggregate view across all database partitions for all active databases in the same instance that contains the currently connected database.

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME, DB_STATUS, SERVER_PLATFORM,
   DB_LOCATION, DB_CONN_TIME
   FROM TABLE(SNAP_GET_DB_V95(CAST (NULL AS VARCHAR(128)), -2)) AS T
```

The following is an example of output from this query.

```
DB_NAME      DB_STATUS    SERVER_PLATFORM ...
--------...- ------------ --------------- ...
TOOLSDB      ACTIVE       AIX64           ...
SAMPLE       ACTIVE       AIX64           ...
```

Output from this query (continued).

```
... DB_LOCATION  DB_CONN_TIME
... ------------ --------------------------
... LOCAL        2005-07-24-22.26.54.396335
... LOCAL        2005-07-24-22.09.22.013196
```

Example 3: This routine can be used by calling the following on the command line:

When connected to a database:

```
SELECT TOTAL_OLAP_FUNCS, OLAP_FUNC_OVERFLOWS, ACTIVE_OLAP_FUNCS
   FROM TABLE (SNAP_GET_DB_V95('', 0)) AS T

The output will look like:

TOTAL_OLAP_FUNCS  OLAP_FUNC_OVERFLOWS  ACTIVE_OLAP_FUNCS
```

```
---------------  ------------------- -------------------
              7                   2                   1

1 record(s) selected.
```

Example 4: After running a workload, a user can use the following query with the table function.

```
 SELECT STATS_CACHE_SIZE, STATS_FABRICATIONS, SYNC_RUNSTATS,
   ASYNC_RUNSTATS, STATS_FABRICATE_TIME, SYNC_RUNSTATS_TIME
   FROM TABLE (SNAP_GET_DB_V95('mytestdb', -1)) AS SNAPDB

STATS_CACHE_SIZE STATS_FABRICATIONS SYNC_RUNSTATS ASYNC_RUNSTATS ...
---------------- ------------------ ------------- -------------- ...
             200                  1             2              0 ...

Continued

...STATS_FABRICATE_TIME  SYNC_RUNSTATS_TIME
...---------------------  ------------------
...                    2                  32

  1 record(s) selected.
```

SNAPDB administrative view and SNAP_GET_DB_V95 table function metadata

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
DB_PATH	VARCHAR(1024)	db_path - Database path
INPUT_DB_ALIAS	VARCHAR(128)	input_db_alias - Input database alias
DB_STATUS	VARCHAR(12)	db_status - Status of database. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • ACTIVE • QUIESCE_PEND • QUIESCED • ROLLFWD • ACTIVE_STANDBY • STANDBY
CATALOG_PARTITION	SMALLINT	catalog_node - Catalog node number
CATALOG_PARTITION_NAME	VARCHAR(128)	catalog_node_name - Catalog node network name

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
SERVER_PLATFORM	VARCHAR(12)	server_platform - Server operating system. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • AIX • AIX64 • AS400_DRDA • DOS • DYNIX • HP • HP64 • HPIA • HPIA64 • LINUX • LINUX390 • LINUXIA64 • LINUXPPC • LINUXPPC64 • LINUXX8664 • LINUXZ64 • MAC • MVS_DRDA • NT • NT64 • OS2 • OS390 • SCO • SGI • SNI • SUN • SUN64 • UNKNOWN • UNKNOWN_DRDA • VM_DRDA • VSE_DRDA • WINDOWS
DB_LOCATION	VARCHAR(12)	db_location - Database location. This interface returns a text identifier based on defines in `sqlmon.h`, and is one of: • LOCAL • REMOTE
DB_CONN_TIME	TIMESTAMP	db_conn_time - Database activation timestamp
LAST_RESET	TIMESTAMP	last_reset - Last reset timestamp
LAST_BACKUP	TIMESTAMP	last_backup - Last backup timestamp

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
CONNECTIONS_TOP	BIGINT	connections_top - Maximum number of concurrent connections
TOTAL_CONS	BIGINT	total_cons - Connects since database activation
TOTAL_SEC_CONS	BIGINT	total_sec_cons - Secondary connections
APPLS_CUR_CONS	BIGINT	appls_cur_cons - Applications connected currently
APPLS_IN_DB2	BIGINT	appls_in_db2 - Applications executing in the database currently
NUM_ASSOC_AGENTS	BIGINT	num_assoc_agents - Number of associated agents
AGENTS_TOP	BIGINT	agents_top - Number of agents created
COORD_AGENTS_TOP	BIGINT	coord_agents_top - Maximum number of coordinating agents
LOCKS_HELD	BIGINT	locks_held - Locks held
LOCK_WAITS	BIGINT	lock_waits - Lock waits
LOCK_WAIT_TIME	BIGINT	lock_wait_time - Time waited on locks
LOCK_LIST_IN_USE	BIGINT	lock_list_in_use - Total lock list memory in use
DEADLOCKS	BIGINT	deadlocks - Deadlocks detected
LOCK_ESCALS	BIGINT	lock_escals - Number of lock escalations
X_LOCK_ESCALS	BIGINT	x_lock_escals - Exclusive lock escalations
LOCKS_WAITING	BIGINT	locks_waiting - Current agents waiting on locks
LOCK_TIMEOUTS	BIGINT	lock_timeouts - Number of lock timeouts
NUM_INDOUBT_TRANS	BIGINT	num_indoubt_trans - Number of indoubt transactions
SORT_HEAP_ALLOCATED	BIGINT	sort_heap_allocated - Total sort heap allocated
SORT_SHRHEAP_ALLOCATED	BIGINT	sort_shrheap_allocated - Sort share heap currently allocated
SORT_SHRHEAP_TOP	BIGINT	sort_shrheap_top - Sort share heap high water mark
POST_SHRTHRESHOLD_SORTS	BIGINT	post_shrthreshold_sorts - Post shared threshold sorts
TOTAL_SORTS	BIGINT	total_sorts - Total sorts
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
ACTIVE_SORTS	BIGINT	active_sorts - Active sorts
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_WRITES	BIGINT	pool_xda_writes - Buffer Pool XDA Data Writes
POOL_ASYNC_XDA_READS	BIGINT	pool_async_xda_reads - Buffer Pool Asynchronous XDA Data Reads
POOL_ASYNC_XDA_WRITES	BIGINT	pool_async_xda_writes - Buffer Pool Asynchronous XDA Data Writes
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_ASYNC_XDA_ READ_REQS	BIGINT	pool_async_xda_read_reqs - Buffer Pool Asynchronous XDA Read Requests
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
POOL_LSN_GAP_CLNS	BIGINT	pool_lsn_gap_clns - Buffer pool log space cleaners triggered
POOL_DRTY_PG_STEAL_CLNS	BIGINT	pool_drty_pg_steal_clns - Buffer pool victim page cleaners triggered
POOL_DRTY_PG_THRSH_CLNS	BIGINT	pool_drty_pg_thrsh_clns - Buffer pool threshold cleaners triggered
PREFETCH_WAIT_TIME	BIGINT	prefetch_wait_time - Time waited for prefetch
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time
FILES_CLOSED	BIGINT	files_closed - Database files closed
ELAPSED_EXEC_TIME_S	BIGINT	elapsed_exec_time - Statement execution elapsed time (in seconds)*
ELAPSED_EXEC_TIME_MS	BIGINT	elapsed_exec_time - Statement execution elapsed time (fractional, in microseconds)*
COMMIT_SQL_STMTS	BIGINT	commit_sql_stmts - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	rollback_sql_stmts - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	dynamic_sql_stmts - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	static_sql_stmts - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	failed_sql_stmts - Failed statement operations
SELECT_SQL_STMTS	BIGINT	select_sql_stmts - Select SQL statements executed

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
UID_SQL_STMTS	BIGINT	uid_sql_stmts - UPDATE/INSERT/DELETE SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements
INT_AUTO_REBINDS	BIGINT	int_auto_rebinds - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
INT_COMMITS	BIGINT	int_commits - Internal commits
INT_ROLLBACKS	BIGINT	int_rollbacks - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	int_deadlock_rollbacks - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_SELECTED	BIGINT	rows_selected - Rows selected
ROWS_READ	BIGINT	rows_read - Rows read
BINDS_PRECOMPILES	BIGINT	binds_precompiles - Binds/precompiles attempted
TOTAL_LOG_AVAILABLE	BIGINT	total_log_available - Total log available
TOTAL_LOG_USED	BIGINT	total_log_used - Total log space used
SEC_LOG_USED_TOP	BIGINT	sec_log_used_top - Maximum secondary log space used
TOT_LOG_USED_TOP	BIGINT	tot_log_used_top - Maximum total log space used
SEC_LOGS_ALLOCATED	BIGINT	sec_logs_allocated - Secondary logs allocated currently
LOG_READS	BIGINT	log_reads - Number of log pages read
LOG_READ_TIME_S	BIGINT	log_read_time - Log read time (in seconds)†
LOG_READ_TIME_NS	BIGINT	log_read_time - Log read time (fractional, in nanoseconds)†
LOG_WRITES	BIGINT	log_writes - Number of log pages written
LOG_WRITE_TIME_S	BIGINT	log_write_time - Log write time (in seconds)†
LOG_WRITE_TIME_NS	BIGINT	log_write_time - Log write time (fractional, in nanoseconds)†
NUM_LOG_WRITE_IO	BIGINT	num_log_write_io - Number of log writes
NUM_LOG_READ_IO	BIGINT	num_log_read_io - Number of log reads
NUM_LOG_PART_PAGE_IO	BIGINT	num_log_part_page_io - Number of partial log page writes
NUM_LOG_BUFFER_FULL	BIGINT	num_log_buffer_full - Number of full log buffers

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
NUM_LOG_DATA_FOUND_IN_BUFFER	BIGINT	num_log_data_found_in_buffer - Number of log data found in buffer
APPL_ID_OLDEST_XACT	BIGINT	appl_id_oldest_xact - Application with oldest transaction
LOG_TO_REDO_FOR_RECOVERY	BIGINT	log_to_redo_for_recovery - Amount of log to be redone for recovery
LOG_HELD_BY_DIRTY_PAGES	BIGINT	log_held_by_dirty_pages - Amount of log space accounted for by dirty pages
PKG_CACHE_LOOKUPS	BIGINT	pkg_cache_lookups - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	pkg_cache_inserts - Package cache inserts
PKG_CACHE_NUM_OVERFLOWS	BIGINT	pkg_cache_num_overflows - Package cache overflows
PKG_CACHE_SIZE_TOP	BIGINT	pkg_cache_size_top - Package cache high water mark
APPL_SECTION_LOOKUPS	BIGINT	appl_section_lookups - Section lookups
APPL_SECTION_INSERTS	BIGINT	appl_section_inserts - Section inserts
CAT_CACHE_LOOKUPS	BIGINT	cat_cache_lookups - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	cat_cache_inserts - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	cat_cache_overflows - Catalog cache overflows
CAT_CACHE_SIZE_TOP	BIGINT	cat_cache_size_top - Catalog cache high water mark
PRIV_WORKSPACE_SIZE_TOP	BIGINT	priv_workspace_size_top - Maximum private workspace size
PRIV_WORKSPACE_NUM_OVERFLOWS	BIGINT	priv_workspace_num_overflows - Private workspace overflows
PRIV_WORKSPACE_SECTION_INSERTS	BIGINT	priv_workspace_section_inserts - Private workspace section inserts
PRIV_WORKSPACE_SECTION_LOOKUPS	BIGINT	priv_workspace_section_lookups - Private workspace section lookups
SHR_WORKSPACE_SIZE_TOP	BIGINT	shr_workspace_size_top - Maximum shared workspace size
SHR_WORKSPACE_NUM_OVERFLOWS	BIGINT	shr_workspace_num_overflows - Shared workspace overflows
SHR_WORKSPACE_SECTION_INSERTS	BIGINT	shr_workspace_section_inserts - Shared workspace section inserts

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
SHR_WORKSPACE_SECTION_LOOKUPS	BIGINT	shr_workspace_section_lookups - Shared workspace section lookups
TOTAL_HASH_JOINS	BIGINT	total_hash_joins - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	hash_join_overflows - Hash join overflows
HASH_JOIN_SMALL_OVERFLOWS	BIGINT	hash_join_small_overflows - Hash join small overflows
POST_SHRTHRESHOLD_HASH_JOINS	BIGINT	post_shrthreshold_hash_joins - Post threshold hash joins
ACTIVE_HASH_JOINS	BIGINT	active_hash_joins - Active hash joins
NUM_DB_STORAGE_PATHS	BIGINT	num_db_storage_paths - Number of automatic storage paths
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
SMALLEST_LOG_AVAIL_NODE	INTEGER	smallest_log_avail_node - Node with least available log space
TOTAL_OLAP_FUNCS	BIGINT	The total number of OLAP functions executed.
OLAP_FUNC_OVERFLOWS	BIGINT	The number of times that OLAP function data exceeded the available sort heap space.
ACTIVE_OLAP_FUNCS	BIGINT	The total number of OLAP functions that are currently running and consuming sort heap memory.
STATS_CACHE_SIZE	BIGINT	The size of the statistics cache in bytes
STATS_FABRICATIONS	BIGINT	Total number of statistics-collect activities for creating statistics by the system without table or index scan.
SYNC_RUNSTATS	BIGINT	Total number of synchronous statistics-collect activities during query compilation.
ASYNC_RUNSTATS	BIGINT	We will change the output for this column to total number of successful asynchronous statistics-collect activities.
STATS_FABRICATE_TIME	BIGINT	Total time spent on creating statistics by system without table or index scan during query compilation in milliseconds.
SYNC_RUNSTATS_TIME	BIGINT	Total time spent on synchronous statistics-collect activities in milliseconds.
NUM_THRESHOLD_VIOLATIONS	BIGINT	The number of threshold violations that have occurred at the database.

Table 298. Information returned by the SNAPDB administrative view and SNAP_GET_DB_V95 table function (continued)

Column name	Data type	Description or corresponding monitor element
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000. †To calculate the total elapsed time for this monitor element, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000,000 + *monitor-element-name*_MS) ÷ 1,000,000,000. For example, (LOG_READ_TIME_S × 1,000,000,000 + LOG_READ_TIME_MS) ÷ 1,000,000,000.		

SNAP_GET_DYN_SQL_V91 table function - Retrieve dynsql logical group snapshot information

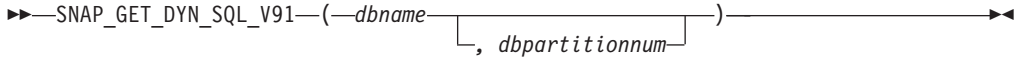
Note: This table function has been deprecated and replaced by the "SNAPDYN_SQL administrative view and SNAP_GET_DYN_SQL_V95 table function - Retrieve dynsql logical group snapshot information" on page 635.

The SNAP_GET_DYN_SQL_V91 table function returns snapshot information from the dynsql logical data group.

This table function returns information equivalent to the GET SNAPSHOT FOR DYNAMIC SQL ON database-alias CLP command.

Refer to Table 299 on page 1129 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify `NULL` or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all

active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_DYN_SQL_V91 table function takes a snapshot for the currently connected database and database partition number.

Authorization

- SYSMON authority
- EXECUTE privilege on the SNAP_GET_DYN_SQL_V91 table function.

Example

Retrieve a list of dynamic SQL run on the currently connected database partition of the currently connected database, ordered by the number of rows read.

```
SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT, 1, 60)
   AS STMT_TEXT FROM TABLE(SNAP_GET_DYN_SQL_V91('',-1)) as T
    ORDER BY ROWS_READ
```

The following is an example of output from this query.

```
PREP_TIME_WORST      ...
-------------------- ...
                   0 ...
                   3 ...
                     ...
                   4 ...
                     ...
                   4 ...
                     ...
                   4 ...
                     ...
                   3 ...
                     ...
                   4 ...
                     ...
```

Output from this query (continued).

```
... NUM_COMPILATIONS     STMT_TEXT
... -------------------- ----------------------------------------...-
...                    0 SET CURRENT LOCALE LC_CTYPE = 'en_US'
...                    1 select rows_read, rows_written,
...                          substr(stmt_text, 1, 40) as
...                    1 select * from table
...                          (snap_get_dyn_sqlv9('',-1)) as t
...                    1 select * from table
...                          (snap_getdetaillog9('',-1)) as t
...                    1 select * from table
...                          (snap_get_hadr('',-1)) as t
...                    1 select prep_time_worst, num_compilations,
...                          substr(stmt_text,
...                    1 select prep_time_worst, num_compilations,
...                          substr(stmt_text,
```

Information returned

Table 299. Information returned by the SNAP_GET_DYN_SQL_V91 table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
NUM_EXECUTIONS	BIGINT	num_executions - Statement executions
NUM_COMPILATIONS	BIGINT	num_compilations - Statement compilations
PREP_TIME_WORST	BIGINT	prep_time_worst - Statement worst preparation time
PREP_TIME_BEST	BIGINT	prep_time_best - Statement best preparation time
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
STMT_SORTS	BIGINT	stmt_sorts - Statement sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_XDA_L_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
POOL_XDA_P_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads

Table 299. Information returned by the SNAP_GET_DYN_SQL_V91 table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_TEMP_XDA_L_READS	BIGINT	pool_temp_xda_l_reads - Buffer Pool Temporary XDA Data Logical Reads
POOL_TEMP_XDA_P_READS	BIGINT	pool_temp_xda_p_reads - Buffer Pool Temporary XDA Data Physical Reads monitor element
TOTAL_EXEC_TIME	BIGINT	total_exec_time - Elapsed statement execution time (in seconds)*
TOTAL_EXEC_TIME_MS	BIGINT	total_exec_time - Elapsed statement execution time (fractional, in microseconds)*
TOTAL_USR_CPU_TIME	BIGINT	total_usr_cpu_time - Total user CPU for a statement (in seconds)*
TOTAL_USR_CPU_TIME_MS	BIGINT	total_usr_cpu_time - Total user CPU for a statement (fractional, in microseconds)*
TOTAL_SYS_CPU_TIME	BIGINT	total_sys_cpu_time - Total system CPU for a statement (in seconds)*
TOTAL_SYS_CPU_TIME_MS	BIGINT	total_sys_cpu_time - Total system CPU for a statement (fractional, in microseconds)*
STMT_TEXT	CLOB(2 M)	stmt_text - SQL statement text
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAP_GET_DYN_SQL

Note: This table function has been deprecated and replaced by the "SNAP_GET_DYN_SQL_V91 table function - Retrieve dynsql logical group snapshot information" on page 1127

The schema is SYSPROC.

The SNAP_GET_DYN_SQL table function returns snapshot information from the dynsql logical data group.

dbname
An input argument of type VARCHAR(255) that specifies a valid database

name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 300. Information returned by the SNAP_GET_DYN_SQL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
NUM_EXECUTIONS	BIGINT	num_executions - Statement executions
NUM_COMPILATIONS	BIGINT	num_compilations - Statement compilations
PREP_TIME_WORST	BIGINT	prep_time_worst - Statement worst preparation time
PREP_TIME_BEST	BIGINT	prep_time_best - Statement best preparation time
INT_ROWS_DELETED	BIGINT	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	int_rows_inserted - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	int_rows_updated - Internal rows updated
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
STMT_SORTS	BIGINT	stmt_sorts - Statement sorts
SORT_OVERFLOWS	BIGINT	sort_overflows - Sort overflows
TOTAL_SORT_TIME	BIGINT	total_sort_time - Total sort time
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads

Table 300. Information returned by the SNAP_GET_DYN_SQL table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
TOTAL_EXEC_TIME	BIGINT	total_exec_time - Elapsed statement execution time (in seconds)*
TOTAL_EXEC_TIME_MS	BIGINT	total_exec_time - Elapsed statement execution time (fractional, in microseconds)*
TOTAL_USR_TIME	BIGINT	total_usr_cpu_time - Total user CPU for a statement (in seconds)*
TOTAL_USR_TIME_MS	BIGINT	total_usr_cpu_time - Total user CPU for a statement (fractional, in microseconds)*
TOTAL_SYS_TIME	BIGINT	total_sys_cpu_time - Total system CPU for a statement (in seconds)*
TOTAL_SYS_TIME_MS	BIGINT	total_sys_cpu_time - Total system CPU for a statement (fractional, in microseconds)*
STMT_TEXT	CLOB	stmt_text - SQL statement text
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPLOCK administrative view and SNAP_GET_LOCK table function – Retrieve lock logical data group snapshot information

Note: This administrative view and table function have been deprecated and replaced by the "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415, "MON_GET_LOCKS - list all locks in the currently connected database" on page 442, and "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381.

The SNAPLOCK administrative view and the SNAP_GET_LOCK table function return snapshot information about locks, in particular, the lock logical data group.

SNAPLOCK administrative view

This administrative view allows you to retrieve lock logical data group snapshot information for the currently connected database.

Used with the SNAPLOCKWAIT administrative view, the SNAPLOCK administrative view provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 185 on page 653 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPLOCK administrative view
- CONTROL privilege on the SNAPLOCK administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_LOCK table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock information for the database partition 0 of the currently connected database.

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
   FROM SYSIBMADM.SNAPLOCK WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
AGENT_ID             LOCK_OBJECT_TYPE LOCK_MODE LOCK_STATUS
-------------------- ---------------- --------- -----------
                   7 TABLE            IX        GRNT

  1 record(s) selected.
```

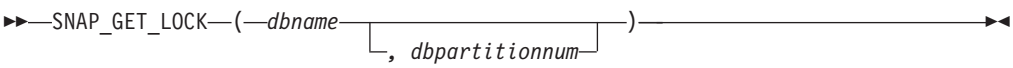
SNAP_GET_LOCK table function

The SNAP_GET_LOCK table function returns the same information as the SNAPLOCK administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_LOCKWAIT table function, the SNAP_GET_LOCK table function provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

Refer to Table 185 on page 653 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify a null value or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for an aggregate of all active database partitions. If *dbname* is not set to `NULL` and *dbpartitionnum* is set to `NULL`, `-1` is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to `NULL`, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_LOCK table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_LOCK table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock information for the current database partition of the currently connected database.

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
   FROM TABLE(SNAP_GET_LOCK('',-1)) as T
```

The following is an example of output from this query.

```
AGENT_ID       LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS
--------...--- ----------------- ---------- -----------
           680 INTERNALV_LOCK     S          GRNT
           680 INTERNALP_LOCK     S          GRNT

2 record(s) selected.
```

Information returned

Table 301. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
TAB_FILE_ID	BIGINT	table_file_id - Table file identification

Table 301. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z

Table 301. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_STATUS	VARCHAR(10)	lock_status - Lock status. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • CONV • GRNT
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
TABNAME	VARCHAR(128)	table_name - Table name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
LOCK_ATTRIBUTES	VARCHAR(128)	lock_attributes - Lock attributes. This interface returns a text identifier based on the defines in `sqlmon.h`. If there are no locks, the text identifier is NONE, otherwise, it is any combination of the following separated by a '+' sign: • ALLOW_NEW • DELETE_IN_BLOCK • ESCALATED • INSERT • NEW_REQUEST • RR • RR_IN_BLOCK • UPDATE_DELETE • WAIT_FOR_AVAIL
LOCK_COUNT	BIGINT	lock_count - Lock count
LOCK_CURRENT_MODE	VARCHAR(10)	lock_current_mode - Original lock mode before conversion. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_HOLD_COUNT	BIGINT	lock_hold_count - Lock hold count

Table 301. Information returned by the SNAPLOCK administrative view and the SNAP_GET_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
LOCK_RELEASE_FLAGS	BIGINT	lock_release_flags - Lock release flags
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element is NULL.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function – Retrieve lockwait logical data group snapshot information

Note: This administrative view and table function have been deprecated and replaced by the "MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks" on page 503 and the "MON_GET_APPL_LOCKWAIT - get information about locks for which an application is waiting" on page 415, "MON_GET_LOCKS - list all locks in the currently connected database" on page 442, and "MON_FORMAT_LOCK_NAME - format the internal lock name and return details" on page 381.

The SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function return snapshot information about lock waits, in particular, the lockwait logical data group.

SNAPLOCKWAIT administrative view

This administrative view allows you to retrieve lockwait logical data group snapshot information for the currently connected database.

Used with the SNAPLOCK administrative view, the SNAPLOCKWAIT administrative view provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

The schema is SYSIBMADM.

Refer to Table 186 on page 658 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- SELECT privilege on the SNAPLOCKWAIT administrative view
- CONTROL privilege on the SNAPLOCKWAIT administrative view
- DATAACCESS authority

In addition, one of the following privileges or authorities is also required:

- EXECUTE privilege on the SNAP_GET_LOCKWAIT table function

- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock wait information on database partition 0 for the currently connected database.

```
SELECT AGENT_ID, LOCK_MODE, LOCK_OBJECT_TYPE, AGENT_ID_HOLDING_LK,
   LOCK_MODE_REQUESTED FROM SYSIBMADM.SNAPLOCKWAIT
   WHERE DBPARTITIONNUM = 0
```

The following is an example of output from this query.

```
AGENT_ID     LOCK_MODE LOCK_OBJECT_TYPE ...
--------...- --------- ---------------- ...
           7 IX        TABLE            ...

  1 record(s) selected.
```

Output from this query (continued).

```
... AGENT_ID_HOLDING_LK  LOCK_MODE_REQUESTED
... -------------------- ------------------
...                   12 IS
```

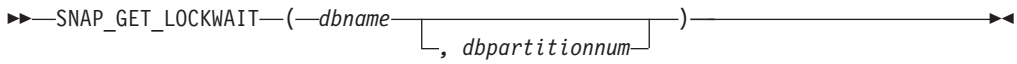
SNAP_GET_LOCKWAIT table function

The SNAP_GET_LOCKWAIT table function returns the same information as the SNAPLOCKWAIT administrative view, but allows you to retrieve the information for a specific database on a specific database partition, aggregate of all database partitions or all database partitions.

Used with the SNAP_GET_LOCK table function, the SNAP_GET_LOCKWAIT table function provides information equivalent to the GET SNAPSHOT FOR LOCKS ON database-alias CLP command.

Refer to Table 186 on page 658 for a complete list of information that can be returned.

Syntax



The schema is SYSPROC.

Table function parameters

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database. Specify a

database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify a null value or empty string to take the snapshot from the currently connected database.

dbpartitionnum
An optional input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for an aggregate of all active database partitions. If *dbname* is not set to NULL and *dbpartitionnum* is set to NULL, -1 is set implicitly for *dbpartitionnum*. If this input option is not used, that is, only *dbname* is provided, data is returned from all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If both *dbname* and *dbpartitionnum* are set to NULL, an attempt is made to read data from the file created by SNAP_WRITE_FILE procedure. Note that this file could have been created at any time, which means that the data might not be current. If a file with the corresponding snapshot API request type does not exist, then the SNAP_GET_LOCKWAIT table function takes a snapshot for the currently connected database and database partition number.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the SNAP_GET_LOCKWAIT table function
- DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:

- SYSMON
- SYSCTRL
- SYSMAINT
- SYSADM

Example

Retrieve lock wait information on current database partition for the currently connected database.

```
SELECT AGENT_ID, LOCK_MODE, LOCK_OBJECT_TYPE, AGENT_ID_HOLDING_LK,
   LOCK_MODE_REQUESTED FROM TABLE(SNAP_GET_LOCKWAIT('',-1)) AS T
```

The following is an example of output from this query.

```
AGENT_ID      LOCK_MODE  LOCK_OBJECT_TYPE   ...
--------...-- ---------- ------------------ ...
          12 X          ROW_LOCK           ...

1 record(s) selected.
```

Output from this query (continued).

```
... AGENT_ID_HOLDING_LK  LOCK_MODE_REQUESTED
... -------------------- ------------------
...                    7 X
```

Usage note

To see lock wait information, you must first turn on the default LOCK monitor switch in the database manager configuration. To have the change take effect immediately explicitly attach to the instance using CLP and then issue the CLP command:

```
UPDATE DATABASE MANAGER CONFIGURATION CLP USING DFT_MON_LOCK ON
```

The default setting can also be turned on through the ADMIN_CMD stored procedure. For example:

```
CALL SYSPROC.ADMIN_CMD('update dbm cfg using DFT_MON_LOCK ON')
```

If the ADMIN_CMD stored procedure is used or if the clp command is used without having previously attached to the instance, the instance must be recycled before the change takes effect.

Information returned

Table 302. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
SUBSECTION_NUMBER	BIGINT	ss_number - Subsection number
LOCK_MODE	VARCHAR(10)	lock_mode - Lock mode. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z

Table 302. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_OBJECT_TYPE	VARCHAR(18)	lock_object_type - Lock object type waited on. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • AUTORESIZE_LOCK • AUTOSTORAGE_LOCK • BLOCK_LOCK • EOT_LOCK • INPLACE_REORG_LOCK • INTERNAL_LOCK • INTERNALB_LOCK • INTERNALC_LOCK • INTERNALJ_LOCK • INTERNALL_LOCK • INTERNALO_LOCK • INTERNALQ_LOCK • INTERNALP_LOCK • INTERNALS_LOCK • INTERNALT_LOCK • INTERNALV_LOCK • KEYVALUE_LOCK • ROW_LOCK • SYSBOOT_LOCK • TABLE_LOCK • TABLE_PART_LOCK • TABLESPACE_LOCK • XML_PATH_LOCK
AGENT_ID_HOLDING_LK	BIGINT	agent_id_holding_lock - Agent ID holding lock
LOCK_WAIT_START_TIME	TIMESTAMP	lock_wait_start_time - Lock wait start timestamp

Table 302. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_MODE_REQUESTED	VARCHAR(10)	lock_mode_requested - Lock mode requested. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_ESCALATION	SMALLINT	lock_escalation - Lock escalation
TABNAME	VARCHAR(128)	table_name - Table name
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
APPL_ID_HOLDING_LK	VARCHAR(128)	appl_id_holding_lk - Application ID holding lock
LOCK_ATTRIBUTES	VARCHAR(128)	lock_attributes - Lock attributes. This interface returns a text identifier based on the defines in `sqlmon.h`. If there are no locks, the text identifier is NONE, otherwise, it is any combination of the following separated by a '+' sign: • ALLOW_NEW • DELETE_IN_BLOCK • ESCALATED • INSERT • NEW_REQUEST • RR • RR_IN_BLOCK • UPDATE_DELETE • WAIT_FOR_AVAIL

Table 302. Information returned by the SNAPLOCKWAIT administrative view and the SNAP_GET_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_CURRENT_MODE	VARCHAR(10)	lock_current_mode - Original lock mode before conversion. This interface returns a text identifier based on the defines in `sqlmon.h` and is one of: • IN • IS • IX • NON (if no lock) • NS • NW • S • SIX • U • X • Z
LOCK_NAME	VARCHAR(32)	lock_name - Lock name
LOCK_RELEASE_FLAGS	BIGINT	lock_release_flags - Lock release flags.
DATA_PARTITION_ID	INTEGER	data_partition_id - Data Partition identifier. For a non-partitioned table, this element is NULL.
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

SNAP_GET_STO_PATHS

Note: This table function has been deprecated and replaced by the "SNAPSTORAGE_PATHS administrative view and SNAP_GET_STORAGE_PATHS table function - Retrieve automatic storage path information" on page 807

The schema is SYSPROC.

The SNAP_GET_STO_PATHS table function returns snapshot information from the storage_paths logical data group.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the `NULL` value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

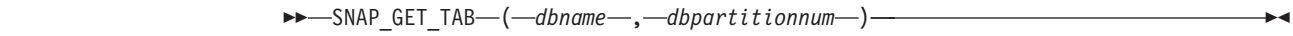
The function returns a table as shown below.

Table 303. Information returned by the SNAP_GET_STO_PATHS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
DB_NAME	VARCHAR(128)	db_name - Database name
DB_STORAGE_PATH	VARCHAR(256)	db_storage_path - Automatic storage path

SNAP_GET_TAB

Note: This table function has been deprecated and replaced by the "SNAPTAB administrative view and SNAP_GET_TAB_V91 table function - Retrieve table logical data group snapshot information" on page 677



The schema is SYSPROC.

The SNAP_GET_TAB table function returns snapshot information from the table logical data group.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the NULL value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 304. Information returned by the SNAP_GET_TAB table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
TAB_FILE_ID	BIGINT	table_file_id - Table file identification
TAB_TYPE	BIGINT	table_type - Table type
DATA_OBJECT_PAGES	BIGINT	data_object_pages - Data object pages
INDEX_OBJECT_PAGES	BIGINT	index_object_pages - Index object pages
LOB_OBJECT_PAGES	BIGINT	lob_object_pages - LOB object pages
LONG_OBJECT_PAGES	BIGINT	long_object_pages - Long object pages
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_WRITTEN	BIGINT	rows_written - Rows written
OVERFLOW_ACCESSES	BIGINT	overflow_accesses - Accesses to overflowed records
PAGE_REORGS	BIGINT	page_reorgs - Page reorganizations
DBPARTITIONNUM	SMALLINT	node_number - Node number

SNAP_GET_TBSP

Note: This table function has been deprecated and replaced by the "SNAPTBSP administrative view and SNAP_GET_TBSP_V91 table function - Retrieve table space logical data group snapshot information" on page 686

```
►►—SNAP_GET_TBSP—(—dbname—,—dbpartitionnum—)————————————————————►◄
```

The schema is SYSPROC.

The SNAP_GET_TBSP table function returns snapshot information from the table space logical data group.

dbname
An input argument of type VARCHAR(255) that specifies a valid database

name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition. If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 305. Information returned by the SNAP_GET_TBSP table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR(128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
TBSP_TYPE	SMALLINT	tablespace_type - Table space type
TBSP_CONTENT_TYPE	SMALLINT	tablespace_content_type - Table space content type
TBSP_PAGE_SIZE	BIGINT	tablespace_page_size - Table space page size
TBSP_EXTENT_SIZE	BIGINT	tablespace_extent_size - Table space extent size
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size
TBSP_CUR_POOL_ID	BIGINT	tablespace_cur_pool_id - Buffer pool currently being used
TBSP_NEXT_POOL_ID	BIGINT	tablespace_next_pool_id - Buffer pool that will be used at next startup
FS_CACHING[1]	SMALLINT	fs_caching - File system caching
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_DATA_P_READS	BIGINT	pool_temp_data_p_reads - Buffer pool temporary data physical reads
POOL_ASYNC_DATA_READS	BIGINT	pool_async_data_reads - Buffer pool asynchronous data reads

Table 305. Information returned by the SNAP_GET_TBSP table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_DATA_WRITES	BIGINT	pool_data_writes - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	pool_async_data_writes - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical reads
POOL_TEMP_INDEX_P_READS	BIGINT	pool_temp_index_p_reads - Buffer pool temporary index physical reads
POOL_ASYNC_INDEX_READS	BIGINT	pool_async_index_reads - Buffer pool asynchronous index reads
POOL_INDEX_WRITES	BIGINT	pool_index_writes - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	pool_async_index_writes - Buffer pool asynchronous index writes
POOL_READ_TIME	BIGINT	pool_read_time - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	pool_write_time - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	pool_async_read_time - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	pool_async_write_time - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	pool_async_data_read_reqs - Buffer pool asynchronous read requests
POOL_ASYNC_INDEX_ READ_REQS	BIGINT	pool_async_index_read_reqs - Buffer pool asynchronous index read requests
POOL_NO_VICTIM_BUFFER	BIGINT	pool_no_victim_buffer - Buffer pool no victim buffers
DIRECT_READS	BIGINT	direct_reads - Direct reads from database
DIRECT_WRITES	BIGINT	direct_writes - Direct writes to database
DIRECT_READ_REQS	BIGINT	direct_read_reqs - Direct read requests
DIRECT_WRITE_REQS	BIGINT	direct_write_reqs - Direct write requests
DIRECT_READ_TIME	BIGINT	direct_read_time - Direct read time
DIRECT_WRITE_TIME	BIGINT	direct_write_time - Direct write time

Table 305. Information returned by the SNAP_GET_TBSP table function (continued)

Column name	Data type	Description or corresponding monitor element
FILES_CLOSED	BIGINT	files_closed - Database files closed
UNREAD_PREFETCH_PAGES	BIGINT	unread_prefetch_pages - Unread prefetch pages
POOL_DATA_TO_ESTORE	BIGINT	The pool_data_to_estore ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_TO_ESTORE	BIGINT	The pool_index_to_estore ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_FROM_ESTORE	BIGINT	The pool_index_from_estore ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_DATA_FROM_ESTORE	BIGINT	The pool_data_from_estore ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
TBSP_REBALANCER_MODE	BIGINT	tablespace_rebalancer_mode - Rebalancer mode
TBSP_USING_AUTO_STORAGE	SMALLINT	tablespace_using_auto_storage - Table space enabled for automatic storage
TBSP_AUTO_RESIZE_ENABLED	SMALLINT	tablespace_auto_resize_enabled - Table space automatic resizing enabled
[1] If FS_CACHING is 0, file system caching is enabled, and if FS_CACHING is 1, file system caching is disabled.		

SNAP_GET_TBSP_PART

Note: This table function has been deprecated and replaced by the "SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V91 table function - Retrieve tablespace_nodeinfo logical data group snapshot information" on page 832

The schema is SYSPROC.

The SNAP_GET_TBSP_PART table function returns snapshot information from the tablespace_nodeinfo logical data group.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling

this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition. If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 306. Information returned by the SNAP_GET_TBSP_PART table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TBSP_NAME	VARCHAR (128)	tablespace_name - Table space name
TBSP_ID	BIGINT	tablespace_id - Table space identification
TBSP_STATE	BIGINT	tablespace_state - Table space state
TBSP_PREFETCH_SIZE	BIGINT	tablespace_prefetch_size - Table space prefetch size
TBSP_NUM_QUIESCERS	BIGINT	tablespace_num_quiescers - Number of quiescers
TBSP_STATE_CHANGE_OBJECT_ID	BIGINT	tablespace_state_change_object_id - State change object identification
TBSP_STATE_CHANGE_TBSP_ID	BIGINT	tablespace_state_change_ts_id - State change table space identification
TBSP_MIN_RECOVERY_TIME	TIMESTAMP	tablespace_min_recovery_time - Minimum recovery time for rollforward
TBSP_TOTAL_PAGES	BIGINT	tablespace_total_pages - Total pages in table space
TBSP_USABLE_PAGES	BIGINT	tablespace_usable_pages - Usable pages in table space
TBSP_USED_PAGES	BIGINT	tablespace_used_pages - Used pages in table space
TBSP_FREE_PAGES	BIGINT	tablespace_free_pages - Free pages in table space
TBSP_PENDING_FREE_PAGES	BIGINT	tablespace_pending_free_pages - Pending free pages in table space
TBSP_PAGE_TOP	BIGINT	tablespace_page_top - Table space high water mark

Table 306. Information returned by the SNAP_GET_TBSP_PART table function (continued)

Column name	Data type	Description or corresponding monitor element
REBALANCER_MODE	BIGINT	tablespace_rebalancer_mode - Rebalancer mode
REBALANCER_EXTENTS_ REMAINING	BIGINT	tablespace_rebalancer_extents_remaining - Total number of extents to be processed by the rebalancer
REBALANCER_EXTENTS_ PROCESSED	BIGINT	tablespace_rebalancer_extents_processed - Number of extents the rebalancer has processed
REBALANCER_PRIORITY	BIGINT	tablespace_rebalancer_priority - Current rebalancer priority
REBALANCER_START_ TIME	TIMESTAMP	tablespace_rebalancer_start_time - Rebalancer start time
REBALANCER_RESTART_ TIME	TIMESTAMP	tablespace_rebalancer_restart_time - Rebalancer restart time
REBALANCER_LAST_ EXTENT_MOVED	BIGINT	tablespace_rebalancer_last_extent_moved - Last extent moved by the rebalancer
TBSP_NUM_RANGES	BIGINT	tablespace_num_ranges - Number of ranges in the table space map
TBSP_NUM_CONTAINERS	BIGINT	tablespace_num_containers - Number of containers in table space
TBSP_INITIAL_SIZE	BIGINT	tablespace_initial_size - Initial table space size
TBSP_CURRENT_SIZE	BIGINT	tablespace_current_size - Current table space size
TBSP_MAX_SIZE	BIGINT	tablespace_max_size - Maximum table space size
TBSP_INCREASE_SIZE	BIGINT	tablespace_increase_size - Increase size in bytes
TBSP_INCREASE_SIZE_ PERCENT	SMALLINT	tablespace_increase_size_percent - Increase size by percent
TBSP_LAST_RESIZE_ TIME	TIMESTAMP	tablespace_last_resize_time - Time of last successful resize
TBSP_LAST_RESIZE_ FAILED	SMALLINT	tablespace_last_resize_failed - Last resize attempt failed
DBPARTITIONNUM	SMALLINT	node_number - Node number

SNAPSHOT_AGENT

Note: This table function has been deprecated and replaced by the "SNAPAGENT administrative view and SNAP_GET_AGENT table function – Retrieve agent logical data group application snapshot information" on page 573.

```
►►—SNAPSHOT_AGENT—(—dbname—,—dbpartitionnum—)———————————————————————►◄
```

The schema is SYSPROC.

The SNAPSHOT_AGENT function returns information about agents from an application snapshot.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

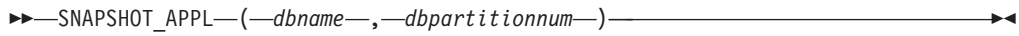
Table 307. Information returned by the SNAPSHOT_AGENT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	agent_id - Application handle (agent ID)
AGENT_PID	BIGINT	agent_pid - Engine dispatchable unit (EDU)

SNAPSHOT_APPL

Returns general information from an application snapshot.

Note: This table function has been deprecated and replaced by the "SNAP_GET_APPL table function – Retrieve appl logical data group snapshot information" on page 1077.



The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 308. Information returned by the SNAPSHOT_APPL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	**agent_id** - Application handle (agent ID)
UOW_LOG_SPACE_USED	BIGINT	**uow_log_space_used** - Unit of work log space used
ROWS_READ	BIGINT	**rows_read** - Rows read
ROWS_WRITTEN	BIGINT	**rows_written** - Rows written
POOL_DATA_L_READS	BIGINT	**pool_data_l_reads** - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	**pool_data_p_reads** - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	**pool_data_writes** - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	**pool_index_l_reads** - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	**pool_index_p_reads** - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	**pool_index_writes** - Buffer pool index writes
POOL_READ_TIME	BIGINT	**pool_read_time** - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	**pool_write_time** - Total buffer pool physical write time

Table 308. Information returned by the SNAPSHOT_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
DIRECT_READS	BIGINT	**direct_reads** - Direct reads from database
DIRECT_WRITES	BIGINT	**direct_writes** - Direct writes to database
DIRECT_READ_REQS	BIGINT	**direct_read_reqs** - Direct read requests
DIRECT_WRITE_REQS	BIGINT	**direct_write_reqs** - Direct write requests
DIRECT_READ_TIME	BIGINT	**direct_read_time** - Direct read time
DIRECT_WRITE_TIME	BIGINT	**direct_write_time** - Direct write time
POOL_DATA_TO_ESTORE	BIGINT	The **pool_data_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_TO_ESTORE	BIGINT	The **pool_index_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_FROM_ESTORE	BIGINT	The **pool_index_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_DATA_FROM_ESTORE	BIGINT	The **pool_data_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
UNREAD_PREFETCH_PAGES	BIGINT	**unread_prefetch_pages** - Unread prefetch pages
LOCKS_HELD	BIGINT	**locks_held** - Locks held
LOCK_WAITS	BIGINT	**lock_waits** - Lock waits
LOCK_WAIT_TIME	BIGINT	**lock_wait_time** - Time waited on locks
LOCK_ESCALS	BIGINT	**lock_escals** - Number of lock escalations
X_LOCK_ESCALS	BIGINT	**x_lock_escals** - Exclusive lock escalations
DEADLOCKS	BIGINT	**deadlocks** - Deadlocks detected
TOTAL_SORTS	BIGINT	**total_sorts** - Total sorts
TOTAL_SORT_TIME	BIGINT	**total_sort_time** - Total sort time
SORT_OVERFLOWS	BIGINT	**sort_overflows** - Sort overflows
COMMIT_SQL_STMTS	BIGINT	**commit_sql_stmts** - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	**rollback_sql_stmts** - Rollback statements attempted

Table 308. Information returned by the SNAPSHOT_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
DYNAMIC_SQL_STMTS	BIGINT	**dynamic_sql_stmts** - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	**static_sql_stmts** - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	**failed_sql_stmts** - Failed statement operations
SELECT_SQL_STMTS	BIGINT	**select_sql_stmts** - Select SQL statements executed
DDL_SQL_STMTS	BIGINT	**ddl_sql_stmts** - Data definition language (DDL) SQL statements
UID_SQL_STMTS	BIGINT	**uid_sql_stmts** - UPDATE/INSERT/DELETE SQL statements executed
INT_AUTO_REBINDS	BIGINT	**int_auto_rebinds** - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	**int_rows_deleted** - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	**int_rows_updated** - Internal rows updated
INT_COMMITS	BIGINT	**int_commits** - Internal commits
INT_ROLLBACKS	BIGINT	**int_rollbacks** - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	**int_deadlock_rollbacks** - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	**rows_deleted** - Rows deleted
ROWS_INSERTED	BIGINT	**rows_inserted** - Rows inserted
ROWS_UPDATED	BIGINT	**rows_updated** - Rows updated
ROWS_SELECTED	BIGINT	**rows_selected** - Rows selected
BINDS_PRECOMPILES	BIGINT	**binds_precompiles** - Binds/precompiles attempted
OPEN_REM_CURS	BIGINT	**open_rem_curs** - Open remote cursors
OPEN_REM_CURS_BLK	BIGINT	**open_rem_curs_blk** - Open remote cursors with blocking
REJ_CURS_BLK	BIGINT	**rej_curs_blk** - Rejected block cursor requests
ACC_CURS_BLK	BIGINT	**acc_curs_blk** - Accepted block cursor requests
SQL_REQS_SINCE_COMMIT	BIGINT	**sql_reqs_since_commit** - SQL requests since last commit
LOCK_TIMEOUTS	BIGINT	**lock_timeouts** - Number of lock timeouts
INT_ROWS_INSERTED	BIGINT	**int_rows_inserted** - Internal rows inserted
OPEN_LOC_CURS	BIGINT	**open_loc_curs** - Open local cursors

Table 308. Information returned by the SNAPSHOT_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
OPEN_LOC_CURS_BLK	BIGINT	**open_loc_curs_blk** - Open local cursors with blocking
PKG_CACHE_LOOKUPS	BIGINT	**pkg_cache_lookups** - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	**pkg_cache_inserts** - Package cache inserts
CAT_CACHE_LOOKUPS	BIGINT	**cat_cache_lookups** - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	**cat_cache_inserts** - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	**cat_cache_overflows** - Catalog cache overflows
CAT_CACHE_HEAP_FULL	BIGINT	**cat_cache_overflows** - Catalog cache overflows
NUM_AGENTS	BIGINT	**num_agents** - Number of agents working on a statement
AGENTS_STOLEN	BIGINT	**agents_stolen** - Stolen agents
ASSOCIATED_AGENTS_TOP	BIGINT	**associated_agents_top** - Maximum number of associated agents
APPL_PRIORITY	BIGINT	**appl_priority** - Application agent priority
APPL_PRIORITY_TYPE	BIGINT	**appl_priority_type** - Application priority type
PREFETCH_WAIT_TIME	BIGINT	**prefetch_wait_time** - Time waited for prefetch
APPL_SECTION_LOOKUPS	BIGINT	**appl_section_lookups** - Section lookups
APPL_SECTION_INSERTS	BIGINT	**appl_section_inserts** - Section inserts
LOCKS_WAITING	BIGINT	**locks_waiting** - agents waiting on locks
TOTAL_HASH_JOINS	BIGINT	**total_hash_joins** - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	**total_hash_loops** - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	**hash_join_overflows** - Hash join overflows
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	**hash_join_small_overflows** - Hash join small overflows
APPL_IDLE_TIME	BIGINT	**appl_idle_time** - Application idle time
UOW_LOCK_WAIT_TIME	BIGINT	**uow_lock_wait_time** - Total time unit of work waited on locks
UOW_COMP_STATUS	BIGINT	**uow_comp_status** - Unit of work completion status

Table 308. Information returned by the SNAPSHOT_APPL table function (continued)

Column name	Data type	Description or corresponding monitor element
AGENT_USR_CPU_TIME_S	BIGINT	**agent_usr_cpu_time** - User CPU time used by agent (in seconds)*
AGENT_USR_CPU_TIME_MS	BIGINT	**agent_usr_cpu_time** - User CPU time used by agent (fractional, in microseconds)*
AGENT_SYS_CPU_TIME_S	BIGINT	**agent_sys_cpu_time** - System CPU time used by agent (in seconds)*
AGENT_SYS_CPU_TIME_MS	BIGINT	**agent_sys_cpu_time** - System CPU time used by agent (fractional, in microseconds)*
APPL_CON_TIME	TIMESTAMP	**appl_con_time** - Connection request start timestamp
CONN_COMPLETE_TIME	TIMESTAMP	**conn_complete_time** - Connection request completion timestamp
LAST_RESET	TIMESTAMP	**last_reset** - Last reset timestamp
UOW_START_TIME	TIMESTAMP	**uow_start_time** - Unit of work start timestamp
UOW_STOP_TIME	TIMESTAMP	**uow_stop_time** - Unit of work stop timestamp
PREV_UOW_STOP_TIME	TIMESTAMP	**prev_uow_stop_time** - Previous unit of work completion timestamp
UOW_ELAPSED_TIME_S	BIGINT	**uow_elapsed_time** - Most recent unit of work elapsed time (in seconds)*
UOW_ELAPSED_TIME_MS	BIGINT	**uow_elapsed_time** - Most recent unit of work elapsed time (fractional, in microseconds)*
ELAPSED_EXEC_TIME_S	BIGINT	**elapsed_exec_time** - Statement execution elapsed time (in seconds)*
ELAPSED_EXEC_TIME_MS	BIGINT	**elapsed_exec_time** - Statement execution elapsed time (fractional, in microseconds)*
INBOUND_COMM_ADDRESS	VARCHAR(32)	**inbound_comm_address** - Inbound communication address
* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.		

SNAPSHOT_APPL_INFO

Returns general information from an application snapshot.

Note: This table function has been deprecated and replaced by the "SNAP_GET_APPL_INFO table function – Retrieve appl_info logical data group snapshot information" on page 1084.

The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 309. Information returned by the SNAPSHOT_APPL_INFO table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	**agent_id** - Application handle (agent ID)
APPL_STATUS	BIGINT	**appl_status** - Application status
CODEPAGE_ID	BIGINT	**codepage_id** - ID of code page used by application
NUM_ASSOC_AGENTS	BIGINT	**num_assoc_agents** - Number of associated agents
COORD_PARTITION_NUM	BIGINT	**coord_node** - Coordinating node
AUTHORITY_LVL	BIGINT	**authority_lvl** - User authorization level
CLIENT_PID	BIGINT	**client_pid** - Client process ID
COORD_AGENT_PID	BIGINT	**coord_agent_pid** - Coordinator agent
STATUS_CHANGE_TIME	TIMESTAMP	**status_change_time** - Application status change time

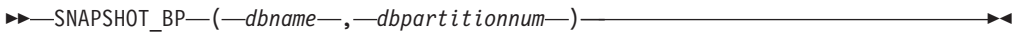
Table 309. Information returned by the SNAPSHOT_APPL_INFO table function (continued)

Column name	Data type	Description or corresponding monitor element
CLIENT_PLATFORM	SMALLINT	**client_platform** - Client operating platform
CLIENT_PROTOCOL	SMALLINT	**client_protocol** - Client communication protocol
COUNTRY_CODE	SMALLINT	**territory_code** - Database territory code
APPL_NAME	VARCHAR(256)	**appl_name** - Application name
APPL_ID	VARCHAR(128)	**appl_id** - Application ID
SEQUENCE_NO	VARCHAR(4)	**sequence_no** - Sequence number
AUTH_ID	VARCHAR(128)	**auth_id** - Authorization ID
CLIENT_NNAME	VARCHAR(128)	The **client_nname** monitor element is deprecated. The value returned is not a valid value.
CLIENT_PRDID	VARCHAR(128)	**client_prdid** - Client product/version ID
INPUT_DB_ALIAS	VARCHAR(128)	**input_db_alias** - Input database alias
CLIENT_DB_ALIAS	VARCHAR(128)	**client_db_alias** - Database alias used by application
DB_NAME	VARCHAR(128)	**db_name** - Database name
DB_PATH	VARCHAR(1024)	**db_path** - Database path
EXECUTION_ID	VARCHAR(128)	**execution_id** - User login ID
CORR_TOKEN	VARCHAR(128)	**corr_token** - DRDA correlation token
TPMON_CLIENT_USERID	VARCHAR(256)	**tpmon_client_userid** - TP monitor client user ID
TPMON_CLIENT_WKSTN	VARCHAR(256)	**tpmon_client_wkstn** - TP monitor client workstation name
TPMON_CLIENT_APP	VARCHAR(256)	**tpmon_client_app** - TP monitor client application name
TPMON_ACC_STR	VARCHAR(200)	**tpmon_acc_str** - TP monitor client accounting string

SNAPSHOT_BP

Returns information from a buffer pool snapshot.

Note: This table function has been deprecated and replaced by the "SNAP_GET_BP table function – Retrieve bufferpool logical group snapshot information" on page 1090.



The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 310. Information returned by the SNAPSHOT_BP table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
POOL_DATA_L_READS	BIGINT	**pool_data_l_reads** - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	**pool_data_p_reads** - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	**pool_data_writes** - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	**pool_index_l_reads** - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	**pool_index_p_reads** - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	**pool_index_writes** - Buffer pool index writes
POOL_READ_TIME	BIGINT	**pool_read_time** - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	**pool_write_time** - Total buffer pool physical write time
POOL_ASYNC_DATA_READS	BIGINT	**pool_async_data_reads** - Buffer pool asynchronous data reads
POOL_ASYNC_DATA_WRITES	BIGINT	**pool_async_data_writes** - Buffer pool asynchronous data writes
POOL_ASYNC_INDEX_WRITES	BIGINT	**pool_async_index_writes** - Buffer pool asynchronous index writes

Table 310. Information returned by the SNAPSHOT_BP table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ASYNC_READ_TIME	BIGINT	**pool_async_read_time** - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	**pool_async_write_time** - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	**pool_async_data_read_reqs** - Buffer pool asynchronous read requests
DIRECT_READS	BIGINT	**direct_reads** - Direct reads from database
DIRECT_WRITES	BIGINT	**direct_writes** - Direct writes to database
DIRECT_READ_REQS	BIGINT	**direct_read_reqs** - Direct read requests
DIRECT_WRITE_REQS	BIGINT	**direct_write_reqs** - Direct write requests
DIRECT_READ_TIME	BIGINT	**direct_read_time** - Direct read time
DIRECT_WRITE_TIME	BIGINT	**direct_write_time** - Direct write time
POOL_ASYNC_INDEX_READS	BIGINT	**pool_async_index_reads** - Buffer pool asynchronous index reads
POOL_DATA_TO_ESTORE	BIGINT	The **pool_data_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_TO_ESTORE	BIGINT	The **pool_index_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_FROM_ESTORE	BIGINT	The **pool_index_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_DATA_FROM_ESTORE	BIGINT	The **pool_data_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
UNREAD_PREFETCH_PAGES	BIGINT	**unread_prefetch_pages** - Unread prefetch pages
FILES_CLOSED	BIGINT	**files_closed** - Database files closed
BP_NAME	VARCHAR(128)	**bp_name** - Buffer pool name
DB_NAME	VARCHAR(128)	**db_name** - Database name
DB_PATH	VARCHAR(1024)	**db_path** - Database path
INPUT_DB_ALIAS	VARCHAR(128)	**input_db_alias** - Input database alias

SNAPSHOT_CONTAINER

Returns container configuration information from a table space snapshot.

Note: This table function has been deprecated and replaced by the "SNAPCONTAINER administrative view and SNAP_GET_CONTAINER_V91 table function - Retrieve tablespace_container logical data group snapshot information" on page 605

The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 311. Information returned by the SNAPSHOT_CONTAINER table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLESPACE_ID	BIGINT	**tablespace_id** - Table space identification
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name
CONTAINER_ID	BIGINT	**container_id** - Container identification
CONTAINER_NAME	VARCHAR(256)	**container_name** - Container name
CONTAINER_TYPE	SMALLINT	**container_type** - Container type
TOTAL_PAGES	BIGINT	**container_total_pages** - Total pages in container

Table 311. Information returned by the SNAPSHOT_CONTAINER table function (continued)

Column name	Data type	Description or corresponding monitor element
USABLE_PAGES	BIGINT	**container_usable_pages** - Usable pages in container
ACCESSIBLE	BIGINT	**container_accessible** - Accessibility of container
STRIPE_SET	BIGINT	**container_stripe_set** - Stripe set

SNAPSHOT_DATABASE

Returns information from a database snapshot.

Note: This table function has been deprecated and replaced by the "SNAP_GET_DB_V91 table function - Retrieve snapshot information from the dbase logical group" on page 1105

The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 312. Information returned by the SNAPSHOT_DATABASE table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
SEC_LOG_USED_TOP	BIGINT	**sec_log_used_top** - Maximum secondary log space used

Table 312. Information returned by the SNAPSHOT_DATABASE table function (continued)

Column name	Data type	Description or corresponding monitor element
TOT_LOG_USED_TOP	BIGINT	**tot_log_used_top** - Maximum total log space used
TOTAL_LOG_USED	BIGINT	**total_log_used** - Total log space used
TOTAL_LOG_AVAILABLE	BIGINT	**total_log_available** - Total log available
ROWS_READ	BIGINT	**rows_read** - Rows read
POOL_DATA_L_READS	BIGINT	**pool_data_l_reads** - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	**pool_data_p_reads** - Buffer pool data physical reads
POOL_DATA_WRITES	BIGINT	**pool_data_writes** - Buffer pool data writes
POOL_INDEX_L_READS	BIGINT	**pool_index_l_reads** - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	**pool_index_p_reads** - Buffer pool index physical reads
POOL_INDEX_WRITES	BIGINT	**pool_index_writes** - Buffer pool index writes
POOL_READ_TIME	BIGINT	**pool_read_time** - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	**pool_write_time** - Total buffer pool physical write time
POOL_ASYNC_INDEX_READS	BIGINT	**pool_async_index_reads** - Buffer pool asynchronous index reads
POOL_DATA_TO_ESTORE	BIGINT	The **pool_data_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_TO_ESTORE	BIGINT	The **pool_index_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_FROM_ESTORE	BIGINT	The **pool_index_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_DATA_FROM_ESTORE	BIGINT	The **pool_data_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_ASYNC_DATA_READS	BIGINT	**pool_async_data_reads** - Buffer pool asynchronous data reads
POOL_ASYNC_DATA_WRITES	BIGINT	**pool_async_data_writes** - Buffer pool asynchronous data writes

Table 312. Information returned by the SNAPSHOT_DATABASE table function (continued)

Column name	Data type	Description or corresponding monitor element
POOL_ASYNC_INDEX_WRITES	BIGINT	**pool_async_index_writes** - Buffer pool asynchronous index writes
POOL_ASYNC_READ_TIME	BIGINT	**pool_async_read_time** - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	**pool_async_write_time** - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	**pool_async_data_read_reqs** - Buffer pool asynchronous read requests
DIRECT_READS	BIGINT	**direct_reads** - Direct reads from database
DIRECT_WRITES	BIGINT	**direct_writes** - Direct writes to database
DIRECT_READ_REQS	BIGINT	**direct_read_reqs** - Direct read requests
DIRECT_WRITE_REQS	BIGINT	**direct_write_reqs** - Direct write requests
DIRECT_READ_TIME	BIGINT	**direct_read_time** - Direct read time
DIRECT_WRITE_TIME	BIGINT	**direct_write_time** - Direct write time
UNREAD_PREFETCH_PAGES	BIGINT	**unread_prefetch_pages** - Unread prefetch pages
FILES_CLOSED	BIGINT	**files_closed** - Database files closed
POOL_LSN_GAP_CLNS	BIGINT	**pool_lsn_gap_clns** - Buffer pool log space cleaners triggered
POOL_DRTY_PG_STEAL_CLNS	BIGINT	**pool_drty_pg_steal_clns** - Buffer pool victim page cleaners triggered
POOL_DRTY_PG_THRSH_CLNS	BIGINT	**pool_drty_pg_thrsh_clns** - Buffer pool threshold cleaners triggered
LOCKS_HELD	BIGINT	**locks_held** - Locks held
LOCK_WAITS	BIGINT	**lock_waits** - Lock waits
LOCK_WAIT_TIME	BIGINT	**lock_wait_time** - Time waited on locks
LOCK_LIST_IN_USE	BIGINT	**lock_list_in_use** - Total lock list memory in use
DEADLOCKS	BIGINT	**deadlocks** - Deadlocks detected
LOCK_ESCALS	BIGINT	**lock_escals** - Number of lock escalations
X_LOCK_ESCALS	BIGINT	**x_lock_escals** - Exclusive lock escalations
LOCKS_WAITING	BIGINT	**locks_waiting** - agents waiting on locks
SORT_HEAP_ALLOCATED	BIGINT	**sort_heap_allocated** - Total sort heap allocated
TOTAL_SORTS	BIGINT	**total_sorts** - Total sorts

Table 312. Information returned by the SNAPSHOT_DATABASE table function (continued)

Column name	Data type	Description or corresponding monitor element
TOTAL_SORT_TIME	BIGINT	**total_sort_time** - Total sort time
SORT_OVERFLOWS	BIGINT	**sort_overflows** - Sort overflows
ACTIVE_SORTS	BIGINT	**active_sorts** - Active sorts
COMMIT_SQL_STMTS	BIGINT	**commit_sql_stmts** - Commit statements attempted
ROLLBACK_SQL_STMTS	BIGINT	**rollback_sql_stmts** - Rollback statements attempted
DYNAMIC_SQL_STMTS	BIGINT	**dynamic_sql_stmts** - Dynamic SQL statements attempted
STATIC_SQL_STMTS	BIGINT	**static_sql_stmts** - Static SQL statements attempted
FAILED_SQL_STMTS	BIGINT	**failed_sql_stmts** - Failed statement operations
SELECT_SQL_STMTS	BIGINT	**select_sql_stmts** - Select SQL statements executed
DDL_SQL_STMTS	BIGINT	ddl_sql_stmts - Data definition language (DDL) SQL statements
UID_SQL_STMTS	BIGINT	**uid_sql_stmts** - UPDATE/INSERT/DELETE SQL statements executed
INT_AUTO_REBINDS	BIGINT	**int_auto_rebinds** - Internal automatic rebinds
INT_ROWS_DELETED	BIGINT	**int_rows_deleted** - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	**int_rows_updated** - Internal rows updated
INT_COMMITS	BIGINT	**int_commits** - Internal commits
INT_ROLLBACKS	BIGINT	**int_rollbacks** - Internal rollbacks
INT_DEADLOCK_ROLLBACKS	BIGINT	**int_deadlock_rollbacks** - Internal rollbacks due to deadlock
ROWS_DELETED	BIGINT	**rows_deleted** - Rows deleted
ROWS_INSERTED	BIGINT	**rows_inserted** - Rows inserted
ROWS_UPDATED	BIGINT	**rows_updated** - Rows updated
ROWS_SELECTED	BIGINT	**rows_selected** - Rows selected
BINDS_PRECOMPILES	BIGINT	**binds_precompiles** - Binds/precompiles attempted
TOTAL_CONS	BIGINT	**total_cons** - Connects since database activation
APPLS_CUR_CONS	BIGINT	**appls_cur_cons** - Applications connected currently
APPLS_IN_DB2	BIGINT	**appls_in_db2** - Applications executing in the database currently
SEC_LOGS_ALLOCATED	BIGINT	**sec_logs_allocated** - Secondary logs allocated currently
DB_STATUS	BIGINT	**db_status** - Status of database

Table 312. Information returned by the SNAPSHOT_DATABASE table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_TIMEOUTS	BIGINT	**lock_timeouts** - Number of lock timeouts
CONNECTIONS_TOP	BIGINT	**connections_top** - Maximum number of concurrent connections
DB_HEAP_TOP	BIGINT	**db_heap_top** - Maximum database heap allocated
INT_ROWS_INSERTED	BIGINT	**int_rows_inserted** - Internal rows inserted
LOG_READS	BIGINT	**log_reads** - Number of log pages read
LOG_WRITES	BIGINT	**log_writes** - Number of log pages written
PKG_CACHE_LOOKUPS	BIGINT	**pkg_cache_lookups** - Package cache lookups
PKG_CACHE_INSERTS	BIGINT	**pkg_cache_inserts** - Package cache inserts
CAT_CACHE_LOOKUPS	BIGINT	**cat_cache_lookups** - Catalog cache lookups
CAT_CACHE_INSERTS	BIGINT	**cat_cache_inserts** - Catalog cache inserts
CAT_CACHE_OVERFLOWS	BIGINT	**cat_cache_overflows** - Catalog cache overflows
CAT_CACHE_HEAP_FULL	BIGINT	**cat_cache_overflows** - Catalog cache overflows
CATALOG_PARTITION	SMALLINT	**catalog_node** - Catalog node number
TOTAL_SEC_CONS	BIGINT	**total_sec_cons** - Secondary connections
NUM_ASSOC_AGENTS	BIGINT	**num_assoc_agents** - Number of associated agents
AGENTS_TOP	BIGINT	**agents_top** - Number of agents created
COORD_AGENTS_TOP	BIGINT	**coord_agents_top** - Maximum number of coordinating agents
PREFETCH_WAIT_TIME	BIGINT	**prefetch_wait_time** - Time waited for prefetch
APPL_SECTION_LOOKUPS	BIGINT	**appl_section_lookups** - Section lookups
APPL_SECTION_INSERTS	BIGINT	**appl_section_inserts** - Section inserts
TOTAL_HASH_JOINS	BIGINT	**total_hash_joins** - Total hash joins
TOTAL_HASH_LOOPS	BIGINT	total_hash_loops - Total hash loops
HASH_JOIN_OVERFLOWS	BIGINT	**hash_join_overflows** - Hash join overflows

Table 312. Information returned by the SNAPSHOT_DATABASE table function (continued)

Column name	Data type	Description or corresponding monitor element
HASH_JOIN_SMALL_ OVERFLOWS	BIGINT	**hash_join_small_overflows** - Hash join small overflows
PKG_CACHE_NUM_ OVERFLOWS	BIGINT	**pkg_cache_num_overflows** - Package cache overflows
PKG_CACHE_SIZE_TOP	BIGINT	**pkg_cache_size_top** - Package cache high water mark
DB_CONN_TIME	TIMESTAMP	**db_conn_time** - Database activation timestamp
SQLM_ELM_LAST_RESET	TIMESTAMP	**last_reset** - Last reset timestamp
SQLM_ELM_LAST_BACKUP	TIMESTAMP	**last_backup** - Last backup timestamp
APPL_CON_TIME	TIMESTAMP	**appl_con_time** - Connection request start timestamp
DB_LOCATION	INTEGER	**db_location** - Database location
SERVER_PLATFORM	INTEGER	**server_platform** - Server operating system
APPL_ID_OLDEST_XACT	BIGINT	**appl_id_oldest_xact** - Application with oldest transaction
CATALOG_PARTITION_NAME	VARCHAR(128)	**catalog_node_name** - Catalog node network name
INPUT_DB_ALIAS	VARCHAR(128)	**input_db_alias** - Input database alias
DB_NAME	VARCHAR(128)	**db_name** - Database name
DB_PATH	VARCHAR(1024)	**db_path** - Database path

SNAPSHOT_DBM

Returns information from a snapshot of the DB2 database manager.

Note: This table function has been deprecated and replaced by the "SNAP_GET_DBM table function – Retrieve the dbm logical grouping snapshot information" on page 1102.

The schema is SYSPROC.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If the null value is specified, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 313. Information returned by the SNAPSHOT_DBM table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
SORT_HEAP_ALLOCATED	BIGINT	**sort_heap_allocated** - Total sort heap allocated
POST_THRESHOLD_SORTS	BIGINT	**post_threshold_sorts** - Post threshold sorts
PIPED_SORTS_REQUESTED	BIGINT	**piped_sorts_requested** - Piped sorts requested
PIPED_SORTS_ACCEPTED	BIGINT	**piped_sorts_accepted** - Piped sorts accepted
REM_CONS_IN	BIGINT	**rem_cons_in** - Remote connections to database manager
REM_CONS_IN_EXEC	BIGINT	**rem_cons_in_exec** - Remote Connections Executing in the Database Manager monitor element
LOCAL_CONS	BIGINT	**local_cons** - Local connections
LOCAL_CONS_IN_EXEC	BIGINT	**local_cons_in_exec** - Local Connections Executing in the Database Manager monitor element
CON_LOCAL_DBASES	BIGINT	**con_local_dbases** - Local databases with current connects
AGENTS_REGISTERED	BIGINT	**agents_registered** - Agents registered
AGENTS_WAITING_ON_TOKEN	BIGINT	**agents_waiting_on_token** - Agents waiting for a token
DB2_STATUS	BIGINT	**db_status** - Status of database
AGENTS_REGISTERED_TOP	BIGINT	**agents_registered_top** - Maximum number of agents registered
AGENTS_WAITING_TOP	BIGINT	**agents_waiting_top** - Maximum number of agents waiting
COMM_PRIVATE_MEM	BIGINT	**comm_private_mem** - Committed private memory
IDLE_AGENTS	BIGINT	**idle_agents** - Number of idle agents
AGENTS_FROM_POOL	BIGINT	**agents_from_pool** - Agents assigned from pool
AGENTS_CREATED_ EMPTY_POOL	BIGINT	**agents_created_empty_pool** - Agents created due to empty agent pool

Table 313. Information returned by the SNAPSHOT_DBM table function (continued)

Column name	Data type	Description or corresponding monitor element
COORD_AGENTS_TOP	BIGINT	**coord_agents_top** - Maximum number of coordinating agents
MAX_AGENT_OVERFLOWS	BIGINT	**max_agent_overflows** - Maximum agent overflows
AGENTS_STOLEN	BIGINT	**agents_stolen** - Stolen agents
GW_TOTAL_CONS	BIGINT	**gw_total_cons** - Total number of attempted connections for DB2 Connect
GW_CUR_CONS	BIGINT	**gw_cur_cons** - Current number of connections for DB2 Connect
GW_CONS_WAIT_HOST	BIGINT	**gw_cons_wait_host** - Number of connections waiting for the host to reply
GW_CONS_WAIT_CLIENT	BIGINT	**gw_cons_wait_client** - Number of connections waiting for the client to send request
POST_THRESHOLD_ HASH_JOINS	BIGINT	**post_threshold_hash_joins** - Hash join threshold
INACTIVE_GW_AGENTS	BIGINT	**idle_agents** - Number of idle agents
NUM_GW_CONN_SWITCHES	BIGINT	**num_gw_conn_switches** - Connection switches
DB2START_TIME	TIMESTAMP	**db2start_time** - Start database manager timestamp
LAST_RESET	TIMESTAMP	**last_reset** - Last reset timestamp

SNAPSHOT_DYN_SQL

Returns information from a dynamic SQL snapshot. It replaces the SQLCACHE_SNAPSHOT function, which is still available for compatibility reasons.

Note: This table function has been deprecated and replaced by the "SNAP_GET_DYN_SQL_V91 table function - Retrieve dynsql logical group snapshot information" on page 1127

The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 314. Information returned by the SNAPSHOT_DYN_SQL table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
ROWS_READ	BIGINT	**rows_read** - Rows read
ROWS_WRITTEN	BIGINT	**rows_written** - Rows written
NUM_EXECUTIONS	BIGINT	**num_executions** - Statement executions
NUM_COMPILATIONS	BIGINT	**num_compilations** - Statement compilations
PREP_TIME_WORST	BIGINT	**prep_time_worst** - Statement worst preparation time
PREP_TIME_BEST	BIGINT	**prep_time_best** - Statement best preparation time
INT_ROWS_DELETED	BIGINT	**int_rows_deleted** - Internal rows deleted
INT_ROWS_INSERTED	BIGINT	**int_rows_inserted** - Internal rows inserted
INT_ROWS_UPDATED	BIGINT	**int_rows_updated** - Internal rows updated
STMT_SORTS	BIGINT	**stmt_sorts** - Statement sorts
TOTAL_EXEC_TIME	BIGINT	**total_exec_time** - Elapsed statement execution time
TOTAL_SYS_CPU_TIME	BIGINT	**total_sys_cpu_time** - Total system CPU for a statement
TOTAL_USR_CPU_TIME	BIGINT	**total_usr_cpu_time** - Total user CPU for a statement
STMT_TEXT	CLOB(16M)[1]	**stmt_text** - SQL statement text

[1] STMT_TEXT is defined as CLOB(16M) to allow for future expansion only. Actual output of the statement text is truncated at 64K.

SNAPSHOT_FCM

Note: This table function has been deprecated and replaced by the "SNAPFCM administrative view and SNAP_GET_FCM table function – Retrieve the fcm logical data group snapshot information" on page 641.

The schema is SYSPROC.

The SNAPSHOT_FCM function returns database manager level information regarding the fast communication manager (FCM).

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

The function returns a table as shown below.

Table 315. Information returned by the SNAPSHOT_FCM table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
BUFF_FREE	BIGINT	**buff_free** - FCM buffers currently free
BUFF_FREE_BOTTOM	BIGINT	**buff_free_bottom** - Minimum FCM Buffers Free
MA_FREE	BIGINT	The **ma_free** monitor element is discontinued. A null value is returned for the discontinued monitor element.
MA_FREE_BOTTOM	BIGINT	The **ma_free_bottom** monitor element is discontinued. A null value is returned for the discontinued monitor element.
CE_FREE	BIGINT	The **ce_free monitor** element is discontinued. A null value is returned for the discontinued monitor element.
CE_FREE_BOTTOM	BIGINT	The **ce_free_bottom** monitor element is discontinued. A null value is returned for the discontinued monitor element.

Table 315. Information returned by the SNAPSHOT_FCM table function (continued)

Column name	Data type	Description or corresponding monitor element
RB_FREE	BIGINT	The **rb_free monitor** element is discontinued. A null value is returned for the discontinued monitor element.
RB_FREE_BOTTOM	BIGINT	The **rb_free_bottom** monitor element is discontinued. A null value is returned for the discontinued monitor element.
PARTITION_NUMBER	SMALLINT	**node_number** - Node number

SNAPSHOT_FCMNODE

Returns information from a snapshot of the fast communication manager in the database manager.

Note: This table function has been deprecated and replaced by the "SNAPFCM_PART administrative view and SNAP_GET_FCM_PART table function – Retrieve the fcm_node logical data group snapshot information" on page 643.

The schema is SYSPROC.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If the null value is specified, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 316. Information returned by the SNAPSHOT_FCMNODE table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
CONNECTION_STATUS	BIGINT	**connection_status** - Connection status
TOTAL_BUFFERS_SENT	BIGINT	**total_buffers_sent** - Total FCM buffers sent

Table 316. Information returned by the SNAPSHOT_FCMNODE table function (continued)

Column name	Data type	Description or corresponding monitor element
TOTAL_BUFFERS_RCVD	BIGINT	**total_buffers_rcvd** - Total FCM buffers received
PARTITION_NUMBER	SMALLINT	**node_number** - Node number

SNAPSHOT_FILEW

Note: This procedure has been deprecated and replaced by the "SNAP_WRITE_FILE procedure" on page 712.

The schema is SYSPROC.

The SNAPSHOT_FILEW procedure writes system snapshot data to a file located in the tmp subdirectory of the instance directory. To execute the SNAPSHOT_FILEW procedure, a user must have SYSADM, SYSCTRL, or SYSMAINT authority. The saved snapshot can be read by users who do not have SYSADM, SYSCTRL, or SYSMAINT authority by passing null values as the inputs to snapshot functions.

requestType
An input argument of type SMALLINT that specifies a valid snapshot request type, as defined in sqlmon.h.

dbname
An input argument of type VARCHAR(128) that specifies a valid database name in the same instance as the currently connected database when calling this procedure. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type SMALLINT that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

Example: Take a snapshot of database manager information by specifying a request type of 1 (which corresponds to SQLMA_DB2), and defaulting to the currently connected database and current database partition.

```
CALL SNAPSHOT_FILEW (1, CAST (NULL AS VARCHAR(128)), CAST (NULL AS SMALLINT))
```

This will result in snapshot data being written to /tmp/SQLMA_DB2.dat in the instance directory on UNIX operating systems or to \tmp\SQLMA_DB2.dat in the instance directory on a Windows operating system.

SNAPSHOT_LOCK

Returns information from a lock snapshot.

Note: This table function has been deprecated and replaced by the "SNAPLOCK administrative view and SNAP_GET_LOCK table function – Retrieve lock logical data group snapshot information" on page 650.

The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 317. Information returned by the SNAPSHOT_LOCK table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	**agent_id** - Application handle (agent ID)
TABLE_FILE_ID	BIGINT	**table_file_id** - Table file identification
LOCK_OBJECT_TYPE	BIGINT	**lock_object_type** - Lock object type waited on
LOCK_MODE	BIGINT	**lock_mode** - Lock mode
LOCK_STATUS	BIGINT	**lock_status** - Lock status
LOCK_OBJECT_NAME	BIGINT	**lock_object_name** - Lock object name
PARTITION_NUMBER	SMALLINT	**node_number** - Node number
LOCK_ESCALATION	SMALLINT	**lock_escalation** - Lock escalation
TABLE_NAME	VARCHAR(128)	**table_name** - Table name
TABLE_SCHEMA	VARCHAR(128)	**table_schema** - Table schema name

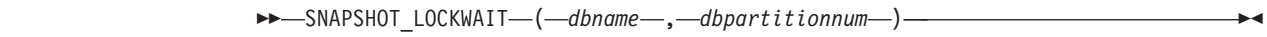
Table 317. Information returned by the SNAPSHOT_LOCK table function (continued)

Column name	Data type	Description or corresponding monitor element
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name

SNAPSHOT_LOCKWAIT

Returns lock waits information from an application snapshot.

Note: This table function has been deprecated and replaced by the "SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function – Retrieve lockwait logical data group snapshot information" on page 655.



The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 318. Information returned by the SNAPSHOT_LOCKWAIT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	**agent_id** - Application handle (agent ID)
SUBSECTION_NUMBER	BIGINT	**ss_number** - Subsection number
LOCK_MODE	BIGINT	**lock_mode** - Lock mode

Table 318. Information returned by the SNAPSHOT_LOCKWAIT table function (continued)

Column name	Data type	Description or corresponding monitor element
LOCK_OBJECT_TYPE	BIGINT	**lock_object_type** - Lock object type waited on
AGENT_ID_HOLDING_LK	BIGINT	**agent_id_holding_lock** - Agent ID holding lock
LOCK_WAIT_START_TIME	TIMESTAMP	**lock_wait_start_time** - Lock wait start timestamp
LOCK_MODE_REQUESTED	BIGINT	**lock_mode_requested** - Lock mode requested
PARTITION_NUMBER	SMALLINT	**node_number** - Node number
LOCK_ESCALATION	SMALLINT	**lock_escalation** - Lock escalation
TABLE_NAME	VARCHAR(128)	**table_name** - Table name
TABLE_SCHEMA	VARCHAR(128)	**table_schema** - Table schema name
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name
APPL_ID_HOLDING_LK	VARCHAR(128)	**appl_id_holding_lk** - Application ID holding lock

SNAPSHOT_QUIESCERS

Note: This table function has been deprecated and replaced by the "SNAPTBSP_QUIESCER administrative view and SNAP_GET_TBSP_QUIESCER table function - Retrieve quiescer table space snapshot information" on page 697.

The schema is SYSPROC.

The SNAPSHOT_QUIESCERS function returns information about quiescers from a table space snapshot.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

The function returns a table as shown below.

Table 319. Information returned by the SNAPSHOT_QUIESCERS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name
QUIESCER_TBS_ID	BIGINT	**quiescer_ts_id** - Quiescer table space identification
QUIESCER_OBJ_ID	BIGINT	**quiescer_obj_id** - Quiescer object identification
QUIESCER_AUTH_ID	BIGINT	**quiescer_auth_id** - Quiescer user authorization identification
QUIESCER_AGENT_ID	BIGINT	**quiescer_agent_id** - Quiescer agent identification
QUIESCER_STATE	BIGINT	**quiescer_state** - Quiescer state

SNAPSHOT_RANGES

Note: This table function has been deprecated and replaced by the "SNAPTBSP_RANGE administrative view and SNAP_GET_TBSP_RANGE table function - Retrieve range snapshot information" on page 701.

The schema is SYSPROC.

The SNAPSHOT_RANGES function returns information from a range snapshot.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

The function returns a table as shown below.

Table 320. Information returned by the SNAPSHOT_RANGES table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLESPACE_ID	BIGINT	**tablespace_id** - Table space identification
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name
RANGE_NUMBER	BIGINT	**range_number** - Range number
RANGE_STRIPE_SET_NUMBER	BIGINT	**range_stripe_set_number** - Stripe set number
RANGE_OFFSET	BIGINT	**range_offset** - Range offset
RANGE_MAX_PAGE	BIGINT	**range_max_page_number** - Maximum page in range
RANGE_MAX_EXTENT	BIGINT	**range_max_extent** - Maximum extent in range
RANGE_START_STRIPE	BIGINT	**range_start_stripe** - Start stripe
RANGE_END_STRIPE	BIGINT	**range_end_stripe** - End stripe
RANGE_ADJUSTMENT	BIGINT	**range_adjustment** - Range adjustment
RANGE_NUM_CONTAINER	BIGINT	**range_num_containers** - Number of containers in range
RANGE_CONTAINER_ID	BIGINT	**range_container_id** - Range container

SNAPSHOT_STATEMENT

Returns information about statements from an application snapshot.

Note: This table function has been deprecated and replaced by the "SNAPSTMT administrative view and SNAP_GET_STMT table function – Retrieve statement snapshot information" on page 661.

The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 321. Information returned by the SNAPSHOT_STATEMENT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
AGENT_ID	BIGINT	**agent_id** - Application handle (agent ID)
ROWS_READ	BIGINT	**rows_read** - Rows read
ROWS_WRITTEN	BIGINT	**rows_written** - Rows written
NUM_AGENTS	BIGINT	**num_agents** - Number of agents working on a statement
AGENTS_TOP	BIGINT	**agents_top** - Number of agents created
STMT_TYPE	BIGINT	**stmt_type** - Statement type
STMT_OPERATION	BIGINT	**stmt_operation/operation** - Statement operation
SECTION_NUMBER	BIGINT	**section_number** - Section number
QUERY_COST_ESTIMATE	BIGINT	**query_cost_estimate** - Query cost estimate
QUERY_CARD_ESTIMATE	BIGINT	**query_card_estimate** - Query number of rows estimate
DEGREE_PARALLELISM	BIGINT	**degree_parallelism** - Degree of parallelism
STMT_SORTS	BIGINT	**stmt_sorts** - Statement sorts
TOTAL_SORT_TIME	BIGINT	**total_sort_time** - Total sort time
SORT_OVERFLOWS	BIGINT	**sort_overflows** - Sort overflows
INT_ROWS_DELETED	BIGINT	**int_rows_deleted** - Internal rows deleted
INT_ROWS_UPDATED	BIGINT	**int_rows_updated** - Internal rows updated
INT_ROWS_INSERTED	BIGINT	**int_rows_inserted** - Internal rows inserted
FETCH_COUNT	BIGINT	**fetch_count** - Number of successful fetches
STMT_START	TIMESTAMP	**stmt_start** - Statement operation start timestamp
STMT_STOP	TIMESTAMP	**stmt_stop** - Statement operation stop timestamp
STMT_USR_CPU_TIME_S	BIGINT	**stmt_usr_cpu_time** - User CPU time used by statement (in seconds)*

Table 321. Information returned by the SNAPSHOT_STATEMENT table function (continued)

Column name	Data type	Description or corresponding monitor element
STMT_USR_CPU_TIME_MS	BIGINT	**stmt_usr_cpu_time** - User CPU time used by statement (fractional, in microseconds)*
STMT_SYS_CPU_TIME_S	BIGINT	**stmt_sys_cpu_time** - System CPU time used by statement (in seconds)*
STMT_SYS_CPU_TIME_MS	BIGINT	**stmt_sys_cpu_time** - System CPU time used by statement (fractional, in microseconds)*
STMT_ELAPSED_TIME_S	BIGINT	**stmt_elapsed_time** - Most recent statement elapsed time (in seconds)*
STMT_ELAPSED_TIME_MS	BIGINT	**stmt_elapsed_time** - Most recent statement elapsed time (fractional, in microseconds)*
BLOCKING_CURSOR	SMALLINT	**blocking_cursor** - Blocking cursor
STMT_PARTITION_NUMBER	SMALLINT	**stmt_node_number** - Statement node
CURSOR_NAME	VARCHAR(128)	**cursor_name** - Cursor name
CREATOR	VARCHAR(128)	**creator** - Application creator
PACKAGE_NAME	VARCHAR(128)	**package_name** - Package name
STMT_TEXT	CLOB(16M)[1]	**stmt_text** - SQL statement text

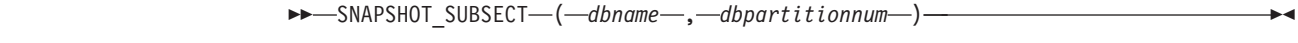
[1] STMT_TEXT is defined as CLOB(16M) to allow for future expansion only. Actual output of the statement text is truncated at 64K.

* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.

SNAPSHOT_SUBSECT

Returns information about subsections of access plans from an application snapshot.

Note: This table function has been deprecated and replaced by the "SNAPSUBSECTION administrative view and SNAP_GET_SUBSECTION table function – Retrieve subsection logical monitor group snapshot information" on page 670.



The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling

this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from all databases under the database instance.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 322. Information returned by the SNAPSHOT_SUBSECT table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
STMT_TEXT	CLOB(16M)[1]	**stmt_text** - SQL statement text
SS_EXEC_TIME	BIGINT	**ss_exec_time** - Subsection execution elapsed time
TQ_TOT_SEND_SPILLS	BIGINT	**tq_tot_send_spills** - Total number of table queue buffers overflowed
TQ_CUR_SEND_SPILLS	BIGINT	**tq_cur_send_spills** - Current number of table queue buffers overflowed
TQ_MAX_SEND_SPILLS	BIGINT	**tq_max_send_spills** - Maximum number of table queue buffers overflows
TQ_ROWS_READ	BIGINT	**tq_rows_read** - Number of rows read from table queues
TQ_ROWS_WRITTEN	BIGINT	**tq_rows_written** - Number of rows written to table queues
ROWS_READ	BIGINT	**rows_read** - Rows read
ROWS_WRITTEN	BIGINT	**rows_written** - Rows written
SS_USR_CPU_TIME	BIGINT	**ss_usr_cpu_time** - User CPU time used by subsection
SS_SYS_CPU_TIME	BIGINT	**ss_sys_cpu_time** - System CPU time used by subsection
SS_NUMBER	INTEGER	**ss_number** - Subsection number
SS_STATUS	INTEGER	**ss_status** - Subsection status
SS_PARTITION_NUMBER	SMALLINT	**ss_node_number** - Subsection node number
TQ_PARTITION_WAITED_FOR	SMALLINT	**tq_node_waited_for** - Waited for node on a table queue

Table 322. Information returned by the SNAPSHOT_SUBSECT table function (continued)

Column name	Data type	Description or corresponding monitor element
TQ_WAIT_FOR_ANY	INTEGER	**tq_wait_for_any** - Waiting for any node to send on a table queue
TQ_ID_WAITING_ON	INTEGER	**tq_id_waiting_on** - Waited on node on a table queue
[1] STMT_TEXT is defined as CLOB(16M) to allow for future expansion only. Actual output of the statement text is truncated at 64K.		

SNAPSHOT_SWITCHES

Returns information about the database snapshot switch state.

Note: This table function has been deprecated and replaced by the "SNAPSWITCHES administrative view and SNAP_GET_SWITCHES table function – Retrieve database snapshot switch state information" on page 674.

The schema is SYSPROC.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

The function returns a table as shown below.

Table 323. Information returned by the SNAPSHOT_SWITCHES table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
UOW_SW_STATE	SMALLINT	State of the unit of work monitor recording switch (0 or 1).
UOW_SW_TIME	TIMESTAMP	If the unit of work monitor recording switch is on, the date and time that this switch was turned on.
STATEMENT_SW_STATE	SMALLINT	State of the SQL statement monitor recording switch (0 or 1).
STATEMENT_SW_TIME	TIMESTAMP	If the SQL statement monitor recording switch is on, the date and time that this switch was turned on.

Table 323. Information returned by the SNAPSHOT_SWITCHES table function (continued)

Column name	Data type	Description or corresponding monitor element
TABLE_SW_STATE	SMALLINT	State of the table activity monitor recording switch (0 or 1).
TABLE_SW_TIME	TIMESTAMP	If the table activity monitor recording switch is on, the date and time that this switch was turned on.
BUFFPOOL_SW_STATE	SMALLINT	State of the buffer pool activity monitor recording switch (0 or 1).
BUFFPOOL_SW_TIME	TIMESTAMP	If the buffer pool activity monitor recording switch is on, the date and time that this switch was turned on.
LOCK_SW_STATE	SMALLINT	State of the lock monitor recording switch (0 or 1).
LOCK_SW_TIME	TIMESTAMP	If the lock monitor recording switch is on, the date and time that this switch was turned on.
SORT_SW_STATE	SMALLINT	State of the sorting monitor recording switch (0 or 1).
SORT_SW_TIME	TIMESTAMP	If the sorting monitor recording switch is on, the date and time that this switch was turned on.
PARTITION_NUMBER	SMALLINT	**node_number** - Node number

SNAPSHOT_TABLE

Returns activity information from a table snapshot.

Note: This table function has been deprecated and replaced by the "SNAPTAB administrative view and SNAP_GET_TAB_V91 table function - Retrieve table logical data group snapshot information" on page 677

The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

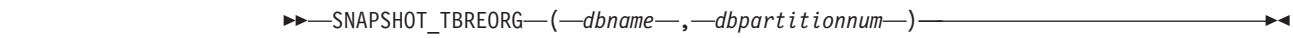
The function returns a table as shown below.

Table 324. Information returned by the SNAPSHOT_TABLE table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
ROWS_WRITTEN	BIGINT	**rows_written** - Rows written
ROWS_READ	BIGINT	**rows_read** - Rows read
OVERFLOW_ACCESSES	BIGINT	**overflow_accesses** - Accesses to overflowed records
TABLE_FILE_ID	BIGINT	**table_file_id** - Table file identification
TABLE_TYPE	BIGINT	**table_type** - Table type
PAGE_REORGS	BIGINT	**page_reorgs** - Page reorganizations
TABLE_NAME	VARCHAR(128)	**table_name** - Table name
TABLE_SCHEMA	VARCHAR(128)	**table_schema** - Table schema name

SNAPSHOT_TBREORG

Note: This table function has been deprecated and replaced by the "SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function - Retrieve table reorganization snapshot information" on page 681.



The schema is SYSPROC.

The SNAPSHOT_TBREORG function returns table reorganization information in the form of a result set. If no tables have been reorganized, 0 rows are returned. To obtain real-time snapshot information, the user must have SYSADM, SYSCTRL, or SYSMAINT authority.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either

"Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

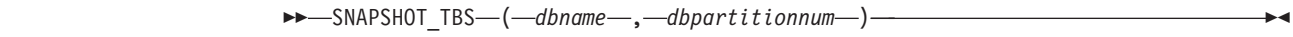
Table 325. Information returned by the SNAPSHOT_TBREORG table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLE_NAME	VARCHAR(128)	**table_name** - Table name
TABLE_SCHEMA	VARCHAR(128)	**table_schema** - Table schema name
PAGE_REORGS	BIGINT	**page_reorgs** - Page reorganizations
REORG_PHASE	BIGINT	**reorg_phase** - Table reorganize phase
REORG_MAX_PHASE	INTEGER	**reorg_max_phase** - Maximum table reorganize phase
REORG_CURRENT_COUNTER	BIGINT	**reorg_current_counter** - Table reorganize progress
REORG_MAX_COUNTER	BIGINT	**reorg_max_counter** - Total amount of table reorganization
REORG_TYPE	INTEGER	**reorg_type** - Table reorganize attributes
REORG_STATUS	BIGINT	**reorg_status** - Table reorganize status
REORG_COMPLETION	INTEGER	**reorg_completion** - Table reorganization completion flag
REORG_START	TIMESTAMP	**reorg_start** - Table reorganize start time
REORG_END	TIMESTAMP	**reorg_end** - Table reorganize end time
REORG_PHASE_START	TIMESTAMP	**reorg_phase_start** - Table reorganize phase start time
REORG_INDEX_ID	BIGINT	**reorg_index_id** - Index used to reorganize the table
REORG_TBSPC_ID	BIGINT	**reorg_tbspc_id** - Table space where table is reorganized
PARTITION_NUMBER	SMALLINT	**node_number** - Node number

SNAPSHOT_TBS

Returns activity information from a table space snapshot.

Note: This table function has been deprecated and replaced by the "SNAPTBSP administrative view and SNAP_GET_TBSP_V91 table function - Retrieve table space logical data group snapshot information" on page 686



The schema is SYSPROC.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify -1 for the current database partition, or -2 for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, -1 is set implicitly.

If both parameters are set to NULL, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 326. Information returned by the SNAPSHOT_TBS table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
POOL_DATA_L_READS	BIGINT	**pool_data_l_reads** - Buffer pool data logical reads
POOL_DATA_P_READS	BIGINT	**pool_data_p_reads** - Buffer pool data physical reads
POOL_ASYNC_DATA_READS	BIGINT	**pool_async_data_reads** - Buffer pool asynchronous data reads
POOL_DATA_WRITES	BIGINT	**pool_data_writes** - Buffer pool data writes
POOL_ASYNC_DATA_WRITES	BIGINT	**pool_async_data_writes** - Buffer pool asynchronous data writes
POOL_INDEX_L_READS	BIGINT	**pool_index_l_reads** - Buffer pool index logical reads
POOL_INDEX_P_READS	BIGINT	**pool_index_p_reads** - Buffer pool index physical reads

Table 326. Information returned by the SNAPSHOT_TBS table function (continued)

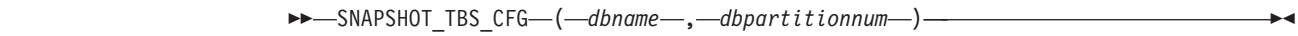
Column name	Data type	Description or corresponding monitor element
POOL_INDEX_WRITES	BIGINT	**pool_index_writes** - Buffer pool index writes
POOL_ASYNC_INDEX_WRITES	BIGINT	**pool_async_index_writes** - Buffer pool asynchronous index writes
POOL_READ_TIME	BIGINT	**pool_read_time** - Total buffer pool physical read time
POOL_WRITE_TIME	BIGINT	**pool_write_time** - Total buffer pool physical write time
POOL_ASYNC_READ_TIME	BIGINT	**pool_async_read_time** - Buffer pool asynchronous read time
POOL_ASYNC_WRITE_TIME	BIGINT	**pool_async_write_time** - Buffer pool asynchronous write time
POOL_ASYNC_DATA_ READ_REQS	BIGINT	**pool_async_data_read_reqs** - Buffer pool asynchronous read requests
DIRECT_READS	BIGINT	**direct_reads** - Direct reads from database
DIRECT_WRITES	BIGINT	**direct_writes** - Direct writes to database
DIRECT_READ_REQS	BIGINT	**direct_read_reqs** - Direct read requests
DIRECT_WRITE_REQS	BIGINT	**direct_write_reqs** - Direct write requests
DIRECT_READ_TIME	BIGINT	**direct_read_time** - Direct read time
DIRECT_WRITE_TIME	BIGINT	**direct_write_time** - Direct write time
UNREAD_PREFETCH_PAGES	BIGINT	**unread_prefetch_pages** - Unread prefetch pages
POOL_ASYNC_INDEX_READS	BIGINT	**pool_async_index_reads** - Buffer pool asynchronous index reads
POOL_DATA_TO_ESTORE	BIGINT	The **pool_data_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_TO_ESTORE	BIGINT	The **pool_index_to_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_INDEX_FROM_ESTORE	BIGINT	The **pool_index_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.
POOL_DATA_FROM_ESTORE	BIGINT	The **pool_data_from_estore** ESTORE monitor element is discontinued. A NULL value is returned for the discontinued monitor element.

Table 326. Information returned by the SNAPSHOT_TBS table function (continued)

Column name	Data type	Description or corresponding monitor element
FILES_CLOSED	BIGINT	**files_closed** - Database files closed
TABLESPACE_NAME	VARCHAR(128)	**tablespace_name** - Table space name

SNAPSHOT_TBS_CFG

Note: This table function has been deprecated and replaced by the "SNAPTBSP_PART administrative view and SNAP_GET_TBSP_PART_V91 table function - Retrieve tablespace_nodeinfo logical data group snapshot information" on page 832



The schema is SYSPROC.

The SNAPSHOT_TBS_CFG function returns configuration information from a table space snapshot.

dbname
An input argument of type VARCHAR(255) that specifies a valid database name in the same instance as the currently connected database when calling this function. Specify a database name that has a directory entry type of either "Indirect" or "Home", as returned by the LIST DATABASE DIRECTORY command. Specify the null value to take the snapshot from the currently connected database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid database partition number. Specify `-1` for the current database partition, or `-2` for all active database partitions. An active database partition is a partition where the database is available for connection and use by applications.

If the null value is specified, `-1` is set implicitly.

If both parameters are set to `NULL`, the snapshot will be taken only if a file has not previously been created by the SNAPSHOT_FILEW stored procedure for the corresponding snapshot API request type.

The function returns a table as shown below.

Table 327. Information returned by the SNAPSHOT_TBS_CFG table function

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	The date and time that the snapshot was taken.
TABLESPACE_ID	BIGINT	**tablespace_id** - Table space identification
TABLESPACE_NAME	VARCHAR (128)	**tablespace_name** - Table space name
TABLESPACE_TYPE	SMALLINT	**tablespace_type** - Table space type

Table 327. Information returned by the SNAPSHOT_TBS_CFG table function (continued)

Column name	Data type	Description or corresponding monitor element
TABLESPACE_STATE	BIGINT	**tablespace_state** - Table space state
NUM_QUIESCERS	BIGINT	**tablespace_num_quiescers** - Number of quiescers
STATE_CHANGE_OBJ_ID	BIGINT	**tablespace_state_change_object_id** - State change object identification
STATE_CHANGE_TBS_ID	BIGINT	**tablespace_state_change_ts_id** - State change table space identification
MIN_RECOVERY_TIME	TIMESTAMP	**tablespace_min_recovery_time** - Minimum recovery time for rollforward
TBS_CONTENTS_TYPE	SMALLINT	**tablespace_content_type** - Table space content type
BUFFERPOOL_ID	BIGINT	**tablespace_cur_pool_id** - Buffer pool currently being used
NEXT_BUFFERPOOL_ID	BIGINT	**tablespace_next_pool_id** - Buffer pool that will be used at next startup
PAGE_SIZE	BIGINT	**tablespace_page_size** - Table space page size
EXTENT_SIZE	BIGINT	**tablespace_extent_size** - Table space extent size
PREFETCH_SIZE	BIGINT	**tablespace_prefetch_size** - Table space prefetch size
TOTAL_PAGES	BIGINT	**tablespace_total_pages** - Total pages in table space
USABLE_PAGES	BIGINT	**tablespace_usable_pages** - Usable pages in table space
USED_PAGES	BIGINT	**tablespace_used_pages** - Used pages in table space
FREE_PAGES	BIGINT	**tablespace_free_pages** - Free pages in table space
PENDING_FREE_PAGES	BIGINT	**tablespace_pending_free_pages** - Pending free pages in table space
HIGH_WATER_MARK	BIGINT	**pool_watermark** - Memory pool watermark
REBALANCER_MODE	BIGINT	**tablespace_rebalancer_mode** - Rebalancer mode
REBALANCER_EXTENTS_REMAINING	BIGINT	**tablespace_rebalancer_extents_remaining** - Total number of extents to be processed by the rebalancer
REBALANCER_EXTENTS_PROCESSED	BIGINT	**tablespace_rebalancer_extents_processed** - Number of extents the rebalancer has processed
REBALANCER_PRIORITY	BIGINT	**tablespace_rebalancer_priority** - Current rebalancer priority
REBALANCER_START_TIME	TIMESTAMP	**tablespace_rebalancer_start_time** - Rebalancer start time

Table 327. Information returned by the SNAPSHOT_TBS_CFG table function (continued)

Column name	Data type	Description or corresponding monitor element
REBALANCER_RESTART_TIME	TIMESTAMP	**tablespace_rebalancer_restart_time** - Rebalancer restart time
LAST_EXTENT_MOVED	BIGINT	**tablespace_rebalancer_last_extent_moved** - Last extent moved by the rebalancer
NUM_RANGES	BIGINT	**tablespace_num_ranges** - Number of ranges in the table space map
NUM_CONTAINERS	BIGINT	**tablespace_num_containers** - Number of containers in table space

SQLCACHE_SNAPSHOT

Note: This table function has been deprecated and replaced by the "SNAP_GET_DYN_SQL_V91 table function - Retrieve dynsql logical group snapshot information" on page 1127

The schema is SYSFUN.

The SQLCACHE_SNAPSHOT function returns the results of a snapshot of the DB2 dynamic SQL statement cache.

The function does not take any arguments. It returns a table, as shown below.

Table 328. Information returned by SQLCACHE_SNAPSHOT table function

Column name	Data type	Description or corresponding monitor element
NUM_EXECUTIONS	INTEGER	num_executions - Statement executions
NUM_COMPILATIONS	INTEGER	num_compilations - Statement compilations
PREP_TIME_WORST	INTEGER	prep_time_worst - Statement worst preparation time
PREP_TIME_BEST	INTEGER	prep_time_best - Statement best preparation time
INT_ROWS_DELETED	INTEGER	int_rows_deleted - Internal rows deleted
INT_ROWS_INSERTED	INTEGER	int_rows_inserted - Internal rows inserted
ROWS_READ	INTEGER	rows_read - Rows read
INT_ROWS_UPDATED	INTEGER	int_rows_updated - Internal rows updated
ROWS_WRITE	INTEGER	rows_written - Rows written
STMT_SORTS	INTEGER	stmt_sorts - Statement sorts

Table 328. Information returned by SQLCACHE_SNAPSHOT table function (continued)

Column name	Data type	Description or corresponding monitor element
TOTAL_EXEC_TIME_S	INTEGER	total_exec_time - Elapsed statement execution time (in seconds)*
TOTAL_EXEC_TIME_MS	INTEGER	total_exec_time - Elapsed statement execution time (fractional, in microseconds)*
TOT_U_CPU_TIME_S	INTEGER	total_usr_cpu_time - Total user CPU for a statement (in seconds)*
TOT_U_CPU_TIME_MS	INTEGER	total_usr_cpu_time - Total user CPU for a statement (fractional, in microseconds)*
TOT_S_CPU_TIME_S	INTEGER	total_sys_cpu_time - Total system CPU for a statement (in seconds)*
TOT_S_CPU_TIME_MS	INTEGER	total_sys_cpu_time - Total system CPU for a statement (fractional, in microseconds)*
DB_NAME	VARCHAR(128)	db_name - Database name
STMT_TEXT	CLOB(16M)[1]	stmt_text - SQL statement text

[1] STMT_TEXT is defined as CLOB(16M) to allow for future expansion only. Actual output of the statement text is truncated at 64K.

* To calculate the total time spent for the monitor element that this column is based on, you must add the full seconds reported in the column for this monitor element that ends with _S to the fractional seconds reported in the column for this monitor element that ends with _MS, using the following formula: (*monitor-element-name*_S × 1,000,000 + *monitor-element-name*_MS) ÷ 1,000,000. For example, (ELAPSED_EXEC_TIME_S × 1,000,000 + ELAPSED_EXEC_TIME_MS) ÷ 1,000,000.

SYSINSTALLROUTINES

Note: This procedure has been deprecated. The procedure was used to create new procedures and functions in DB2 UDB for Linux, UNIX, and Windows Version 8.

The schema is SYSPROC.

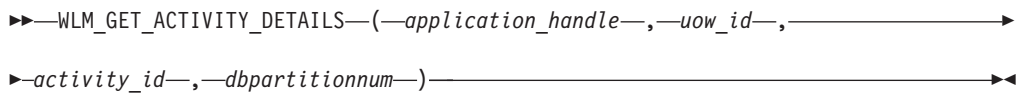
WLM_GET_ACTIVITY_DETAILS - Return detailed information about a specific activity

Note: This table function has been deprecated and replaced by the MON_GET_ACTIVITY_DETAILS table function.

This function returns basic statistics of one or more service subclasses.

This function returns detailed information about a specific activity identified by its application handle, unit of work ID, and activity ID. This information includes details about any thresholds that the activity has violated.

Syntax



The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies a valid application handle. If the argument is null, no rows are returned from this function. If the argument is null, an SQL171N error is returned.

uow_id
An input argument of type INTEGER that specifies a valid unit of work identifier unique within the application. If the argument is null, no rows are returned from this function. If the argument is null, an SQL171N error is returned.

activity_id
An input argument of type INTEGER that specifies a valid activity ID unique within the unit of work. If the argument is null, no rows are returned from this function. If the argument is null, an SQL171N error is returned.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database when calling this function. Specify -1 for the current database partition, or -2 for all database partitions. If a null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_ACTIVITY_DETAILS function.

Example

Detailed information about an individual activity can be obtained by using the WLM_GET_ACTIVITY_DETAILS table function. This table function returns activity information as name-value pairs for each partition. This example is restricted to showing only an eleven member subset of the name-value pairs for each partition for an activity identified by an application handle of 1, a unit of work ID of 1 and an activity ID of 5. For a complete list of name-value pairs, see Table 330 on page 1195 and Table 331 on page 1197.

```
SELECT SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       SUBSTR(NAME, 1, 20) AS NAME,
       SUBSTR(VALUE, 1, 30) AS VALUE
FROM TABLE(WLM_GET_ACTIVITY_DETAILS(1, 1, 5, -2)) AS ACTDETAIL
WHERE NAME IN ('APPLICATION_HANDLE',
               'COORD_PARTITION_NUM',
               'LOCAL_START_TIME',
               'UOW_ID',
               'ACTIVITY_ID',
```

```
              'PARENT_UOW_ID',
              'PARENT_ACTIVITY_ID',
              'ACTIVITY_TYPE',
              'NESTING_LEVEL',
              'INVOCATION_ID',
              'ROUTINE_ID')
  ORDER BY PART
```

The following is an example of output from this query.

```
PART NAME                 VALUE
---- -------------------- ------------------------------
0    APPLICATION_HANDLE   1
0    COORD_PARTITION_NUM  0
0    LOCAL_START_TIME     2005-11-25-18.52.49.343000
0    UOW_ID               1
0    ACTIVITY_ID          5
0    PARENT_UOW_ID        1
0    PARENT_ACTIVITY_ID   3
0    ACTIVITY_TYPE        READ_DML
0    NESTING_LEVEL        0
0    INVOCATION_ID        1
0    ROUTINE_ID           0
1    APPLICATION_HANDLE   1
1    COORD_PARTITION_NUM  0
1    LOCAL_START_TIME     2005-11-25-18.52.49.598000
1    UOW_ID               1
1    ACTIVITY_ID          5
1    PARENT_UOW_ID
1    PARENT_ACTIVITY_ID
1    ACTIVITY_TYPE        READ_DML
1    NESTING_LEVEL        0
1    INVOCATION_ID        1
1    ROUTINE_ID           0
```

Usage note

An ACTIVITY_STATE of QUEUED means that the coordinator activity has made a RPC to the catalog partition to obtain threshold tickets and has not yet received a response. Seeing this state might indicate that the activity has been queued by WLM or, over short periods of time, might just indicate that the activity is in the process of obtaining its tickets. To obtain a more accurate picture of whether or not the activity is really being queued, one can determine which agent is working on the activity (using the WLM_GET_SERVICE_CLASS_AGENTS table function) and find out whether this agent's event_object at the catalog partition has a value of WLM_QUEUE.

Information returned

Table 329. Information returned for WLM_GET_ACTIVITY_DETAILS

Column Name	Data Type	Description
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
NAME	VARCHAR(256)	Element name. See Table 330 on page 1195 and Table 331 on page 1197 for possible values.
VALUE	VARCHAR(1024)	Element values. See Table 330 on page 1195 and Table 331 on page 1197 for possible values.

Table 330. Elements returned

Element Name	Description
ACTIVITY_ID	Unique activity identifier within an application.
ACTIVITY_STATE	Possible values include: • CANCEL_PENDING • EXECUTING • IDLE • INITIALIZING • QP_CANCEL_PENDING • QP_QUEUED • QUEUED • TERMINATING • UNKNOWN
ACTIVITY_TYPE	Possible values include: • CALL • DDL • LOAD • OTHER • READ_DML • WRITE_DML
APPLICATION_HANDLE	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier will be the same on every partition where the application may make a secondary connection.
COORD_PARTITION_NUM	The coordinator partition of the activity.
DATABASE_WORK_ACTION_SET_ID	If this activity has been mapped to a work action set that has been applied to the database, this column contains the ID of the work action set. This column contains 0 if the activity has not been mapped to a work action set that has been applied to the database.
DATABASE_WORK_CLASS_ID	If this activity has been mapped to a work action set that has been applied to the database, this column contains the ID of the work class of this activity. This column contains 0 if the activity has not been mapped to a work action set that has been applied to the database.
EFFECTIVE_ISOLATION	The effective isolation level for this activity.
EFFECTIVE_LOCK_TIMEOUT	The effective lock timeout value for this activity.
EFFECTIVE_QUERY_DEGREE	The effective value of query degree for this activity.
ENTRY_TIME	The time that this activity arrived into the system.
INVOCATION_ID	An identifier that distinguishes one invocation of a routine from others at the same nesting level within a unit of work. It is unique within a unit of work for a specific nesting level.
LAST_REFERENCE_TIME	Every time a request occurs in this activity, this field is updated.

Table 330. Elements returned (continued)

Element Name	Description
LOCAL_START_TIME	The time that this activity began doing work on the partition. It is in local time. This field can be an empty string when an activity has entered the system but is in a queue and has not started executing.
NESTING_LEVEL	This represents the nesting level of this activity. Nesting level is the depth to which this activity is nested within its top-most parent activity.
PACKAGE_NAME	If the activity is a SQL statement, this represents the name of its package.
PACKAGE_SCHEMA	If the activity is a SQL statement, this represents the schema name of its package.
PACKAGE_VERSION_ID	If the activity is a SQL statement, this represents the version of its package.
PARENT_ACTIVITY_ID	Unique activity identifier within a unit of work for the parent of the activity whose ID is ACTIVITY_ID. Returns an empty string if the activity has no parent activity.
PARENT_UOW_ID	Unique unit of work identifier within an application. Refers to the original unit of work this activity's parent activity started in. Returns an empty string if the activity has no parent activity or when at a remote partition.
QP_QUERY_ID	The query ID assigned to this activity by Query Patroller if the activity is a query. A query ID of 0 indicates that Query Patroller did not assign a query ID to this activity.
QUERY_COST_ESTIMATE	Estimated cost, in timerons, for a query, as determined by the SQL compiler.
ROUTINE_ID	Routine unique identifier. Returns zero if the activity is not part of a routine.
ROWS_FETCHED	This is the number of rows read from the table. This reports only those values for the database partition for which this record is recorded. In a partitioned database environment, these values may not reflect the correct totals for the whole activity. When the statement monitor switch is not turned on, this element is not collected and -1 is written instead.
ROWS_MODIFIED	This is the number of rows inserted, updated, or deleted. This reports only those values for the database partition for which this record is recorded. In a partitioned database environment, these values may not reflect the correct totals for the whole activity. When the statement monitor switch is not turned on, this element is not collected and -1 is written instead.
SECTION_NUMBER	If the activity is a SQL statement, this represents its section number.
SERVICE_CLASS_ID	Unique identifier of the service class to which this activity belongs.
SERVICE_CLASS_WORK_ACTION_SET_ID	If this activity has been mapped to a work action set that has been applied to a service class, this column contains the ID of the work action set. This column contains 0 if the activity has not been mapped to a work action set that has been applied to a service class.

Table 330. Elements returned (continued)

Element Name	Description
SERVICE_CLASS_WORK_CLASS_ID	If this activity has been mapped to a work action set that has been applied to a service class, this column contains the ID of the work class of this activity. This column contains 0 if the activity has not been mapped to a work action set that has been applied to a service class.
STMT_PKG_CACHE_ID	Statement package cache identifier.
STMT_TEXT	If the activity is dynamic SQL or it is static SQL for which the statement text is available, this field contains the first 1024 characters of the statement text. It is an empty string otherwise.
SYSTEM_CPU_TIME	The total system CPU time (in seconds and microseconds) used by the database manager agent process, the unit of work, or the statement. When either the statement monitor switch or the timestamp switch is not turned on, this element is not collected and -1 is written instead.
UOW_ID	Unique unit of work identifier within an application. Refers to the original unit of work this activity started in.
USER_CPU_TIME	The total user CPU time (in seconds and microseconds) used by the database manager agent process, the unit of work, or the statement. When either the statement monitor switch or the timestamp switch is not turned on, this element is not collected and -1 is written instead.
UTILITY_ID	If the activity is a utility, this is its utility ID. Otherwise, this field is 0.

Important: The WLM_GET_ACTIVITY_DETAILS table function shows only the thresholds that are currently being applied to an activity.

The following elements are returned only if the corresponding thresholds apply to the activity.

Table 331. Elements returned if applicable

Element Name	Description
ACTIVITYTOTALTIME_THRESHOLD_ID	The ID of the ACTIVITYTOTALTIME threshold that was applied to the activity.
ACTIVITYTOTALTIME_THRESHOLD_VALUE	A timestamp that is computed by adding the ACTIVITYTOTALTIME threshold duration to the activity entry time. If the activity is still executing when this timestamp is reached, the threshold will be violated.
ACTIVITYTOTALTIME_THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the ACTIVITYTOTALTIME threshold. 'No' indicates that the activity has not yet violated the threshold.
CONCURRENTDBCOORDACTIVITIES_DB _THRESHOLD_ID	The ID of the CONCURRENTDBCOORDACTIVITIES_DB threshold that was applied to the activity.
CONCURRENTDBCOORDACTIVITIES_DB _THRESHOLD_QUEUED	'Yes' indicates that the activity was queued by the CONCURRENTDBCOORDACTIVITIES_DB threshold. 'No' indicates that the activity was not queued.

Table 331. Elements returned if applicable (continued)

Element Name	Description
CONCURRENTDBCOORDACTIVITIES_DB _THRESHOLD_VALUE	The upper bound of the CONCURRENTDBCOORDACTIVITIES_DB threshold that was applied to the activity.
CONCURRENTDBCOORDACTIVITIES_DB _THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the CONCURRENTDBCOORDACTIVITIES_DB threshold. 'No' indicates that the activity has not yet violated the threshold.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_ID	The ID of the CONCURRENTDBCOORDACTIVITIES_SUBCLASS threshold that was applied to the activity.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_QUEUED	'Yes' indicates that the activity was queued by the CONCURRENTDBCOORDACTIVITIES_SUBCLASS threshold. 'No' indicates that the activity was not queued.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_VALUE	The upper bound of the CONCURRENTDBCOORDACTIVITIES_SUBCLASS threshold that was applied to the activity.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the CONCURRENTDBCOORDACTIVITIES_SUBCLASS threshold. 'No' indicates that the activity has not yet violated the threshold.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_ID	The ID of the CONCURRENTDBCOORDACTIVITIES _SUPERCLASS threshold that was applied to the activity.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_QUEUED	'Yes' indicates that the activity was queued by the CONCURRENTDBCOORDACTIVITIES _SUPERCLASS threshold. 'No' indicates that the activity was not queued.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_VALUE	The upper bound of the CONCURRENTDBCOORDACTIVITIES _SUPERCLASS threshold that was applied to the activity.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the CONCURRENTDBCOORDACTIVITIES _SUPERCLASS threshold. 'No' indicates that the activity has not yet violated the threshold.
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_ID	The ID of the CONCURRENTDBCOORDACTIVITIES_WORK _ACTION_SET threshold that was applied to the activity.
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_QUEUED	'Yes' indicates that the activity was queued by the CONCURRENTDBCOORDACTIVITIES _WORK_ACTION_SET threshold. 'No' indicates that the activity was not queued.
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_VALUE	The upper bound of the CONCURRENTDBCOORDACTIVITIES_WORK _ACTION_SET threshold that was applied to the activity.

Table 331. Elements returned if applicable (continued)

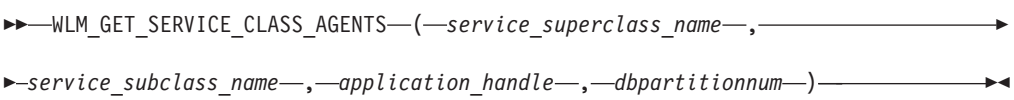
Element Name	Description
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the CONCURRENTDBCOORDACTIVITIES_WORK _ACTION_SET threshold. 'No' indicates that the activity has not yet violated the threshold.
CONCURRENTWORKLOADACTIVITIES _THRESHOLD_ID	The ID of the CONCURRENTWORKLOADACTIVITIES threshold that was applied to the activity.
CONCURRENTWORKLOADACTIVITIES _THRESHOLD_VALUE	The upper bound of the CONCURRENTWORKLOADACTIVITIES threshold that was applied to the activity.
CONCURRENTWORKLOADACTIVITIES _THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the CONCURRENTWORKLOADACTIVITIES threshold. 'No' indicates that the activity has not yet violated the threshold.
ESTIMATEDSQLCOST_THRESHOLD_ID	The ID of the ESTIMATEDSQLCOST threshold that was applied to the activity.
ESTIMATEDSQLCOST_THRESHOLD_VALUE	The upper bound of the ESTIMATEDSQLCOST threshold that was applied to the activity.
ESTIMATEDSQLCOST_THRESHOLD_VIOLATED	'1' indicates that the activity violated the ESTIMATEDSQLCOST threshold. '0' indicates that the activity has not yet violated the threshold.
SQLROWSRETURNED_THRESHOLD_ID	The ID of the SQLROWSRETURNED threshold that was applied to the activity
SQLROWSRETURNED_THRESHOLD_VALUE	The upper bound of the SQLROWSRETURNED threshold that was applied to the activity.
SQLROWSRETURNED_THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the SQLROWSRETURNED threshold. 'No' indicates that the activity has not yet violated the threshold.
SQLTEMPSPACE_THRESHOLD_ID	The ID of the SQLTEMPSPACE threshold that was applied to the activity.
SQLTEMPSPACE_THRESHOLD_VALUE	The upper bound of the SQLTEMPSPACE threshold that was applied to the activity.
SQLTEMPSPACE_THRESHOLD_VIOLATED	'Yes' indicates that the activity violated the SQLTEMPSPACE threshold. 'No' indicates that the activity has not yet violated the threshold.

WLM_GET_SERVICE_CLASS_AGENTS - List agents running in a service class

The WLM_GET_SERVICE_CLASS_AGENTS function returns the list of agents, fenced mode processes (db2fmp processes), and system entities on a specified partition that are running in a specified service class or on behalf of a specified application. The system entities are non-agent threads and processes, such as page cleaners and prefetchers.

Note: This table function has been deprecated and replaced by the "WLM_GET_SERVICE_CLASS_AGENTS_V97 table function - List agents running in a service class" on page 916.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, data is retrieved for all the superclasses in the database that match the values of the other parameters .

service_subclass_name
An input argument of type VARCHAR(128) that refers to a specific subclass within a superclass. If the argument is null or an empty string, data is retrieved for all the subclasses in the database that match the values of the other parameters.

application_handle
An input argument of type BIGINT that specifies the application handle for which agent information is returned. If the argument is null, data is retrieved for all applications in the database that match the values of the other parameters. If you specify 0, only system entities are returned.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If a null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_CLASS_AGENTS function.

Example

The following query returns a list of agents that are associated with application handle 1 for all database partitions. The application handle could have been determined by using the LIST APPLICATIONS command or the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function.

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
     SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
     SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
     SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
     SUBSTR(AGENT_STATE,1,10) AS AGENTSTATE,
     SUBSTR(REQUEST_TYPE,1,12) AS REQTYPE,
     SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
     SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS(CAST(NULL AS VARCHAR(128)),
     CAST(NULL AS VARCHAR(128)), 1, -2)) AS SCDETAILS
ORDER BY APPHANDLE, PART, AGENT_TID
```

Sample output is as follows:

```
APPHANDLE PART AGENT_TID AGENTTYPE   AGENTSTATE REQTYPE        UOW_ID ACT_ID
--------- ---- --------- ----------- ---------- -------------- ------ ------
1         0    3         COORDINATOR ACTIVE     FETCH          1      5
1         0    4         SUBAGENT    ACTIVE     SUBSECTION:1   1      5
1         1    2         SUBAGENT    ACTIVE     SUBSECTION:2   1      5
```

The output shows a coordinator agent and a subagent on partition 0 and a subagent on partition 1 operating on behalf of an activity with UOW ID 1 and activity ID 5. The coordinator agent reports that the request is a fetch request.

Usage note

The parameters are, in effect, ANDed together. That is, if you specify conflicting input parameters, such as a service superclass SUP_A and a subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows are returned.

Information returned

Table 332. Information returned by WLM_GET_SERVICE_CLASS_AGENTS

Column name	Data type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
APPLICATION_HANDLE	BIGINT	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16-bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16-bit counter. In addition, this identifier is the same on every partition where the application makes a secondary connection.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
ENTITY	VARCHAR(32)	One of the following values: • db2agent, which indicates that the type of entity is an agent • db2fmp (*pid*), which indicates that the entity is a fenced mode process, where *pid* is the process ID of the fenced mode process • The name of a system entity
WORKLOAD_NAME	VARCHAR(128)	Name of the workload from which this record was collected.
WORKLOAD_OCCURRENCE_ID	INTEGER	ID of the workload occurrence. This ID does not uniquely identify the workload occurrence unless it is coupled with the coordinator database partition number and the workload name.
UOW_ID	INTEGER	Unique ID of the unit of work that this activity started in.
ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work.
PARENT_UOW_ID	INTEGER	Unique ID of the unit of work that the parent activity of the activity started in. If the activity has no parent, the value of the column is null.
PARENT_ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work for the parent of the activity whose ID is activity_id. If the activity has no parent, the value of the column is null.

Table 332. Information returned by WLM_GET_SERVICE_CLASS_AGENTS (continued)

Column name	Data type	Description
AGENT_TID	BIGINT	Thread ID of the agent or system entity. If this ID is unavailable, the value of the column is null.
AGENT_TYPE	VARCHAR(32)	Agent type. The possible values are as follows: • COORDINATOR • OTHER • PDBSUBAGENT • SMPSUBAGENT If the agent type is COORDINATOR, the agent ID might change in concentrator environments.
SMP_COORDINATOR	INTEGER	Indication of whether the agent is an SMP coordinator. If yes, the value is 1; if no, the value is 0.
AGENT_SUBTYPE	VARCHAR(32)	Agent subtype. The possible values are as follows: • DSS • OTHER • RPC • SMP
AGENT_STATE	VARCHAR(32)	Agent state. The possible values are as follows: • ACTIVE • ASSOCIATED
EVENT_TYPE	VARCHAR(32)	Type of event last processed by this agent. The possible values are as follows: • ACQUIRE • PROCESS • WAIT See Table 333 on page 1204 for more information about possible values for this column.
EVENT_OBJECT	VARCHAR(32)	Object of the event last processed by this agent. The possible values are as follows: • COMPRESSION_DICTIONARY_BUILD • IMPLICIT_REBIND • INDEX_RECREATE • LOCK • LOCK_ESCALATION • QP_QUEUE • REMOTE_REQUEST • REQUEST • ROUTINE • WLM_QUEUE See Table 333 on page 1204 for more information about possible values for this column.

Table 332. Information returned by WLM_GET_SERVICE_CLASS_AGENTS (continued)

Column name	Data type	Description
EVENT_STATE	VARCHAR(32)	State of the event last processed by this agent. The possible values are as follows: • EXECUTING • IDLE See Table 333 on page 1204 for more information about possible values for this column.
REQUEST_ID	VARCHAR(64)	Request ID. This value is unique only in combination with the value of *application_handle*. You can use this to distinguish between one request that is taking a long time and multiple requests; for example, one long fetch and multiple fetches.
REQUEST_TYPE	VARCHAR(32)	Type of request. The possible values are as follows: • For coordinator agents: – CLOSE – COMMIT – COMPILE – DESCRIBE – EXCSQLSET – EXECIMMD – EXECUTE – FETCH – INTERNAL *number*, where *number* is the value of the internal constant – OPEN – PREPARE – REBIND – REDISTRIBUTE – REORG – ROLLBACK – RUNSTATS • For subagents with an AGENT_SUBTYPE of DSS or SMP – If the subsection number is nonzero, the subsection number in the form SUBSECTION:*subsection number*; otherwise, NULL

Table 332. Information returned by WLM_GET_SERVICE_CLASS_AGENTS (continued)

Column name	Data type	Description
REQUEST_TYPE (continued)	VARCHAR(32)	• For subagents with an AGENT_SUBTYPE of RPC – ABP – CATALOG – INTERNAL – REORG – RUNSTATS – WLM • For subagents with an AGENT_SUBTYPE of OTHER" – ABP – APP_RBSVPT – APP_RELSVPT – BACKUP – CLOSE – EXTERNAL_RBSVPT – EVMON – FORCE – FORCE_ALL – INTERNAL *number*, where *number* is the value of the internal constant – INTERRUPT – NOOP (if there is no request) – QP – REDISTRIBUTE – STMT_RBSVPT – STOP_USING – UPDATE_DBM_CFG – WLM
NESTING_LEVEL	INTEGER	Nesting level of the activity whose ID is activity_id. Nesting level is the depth to which this activity is nested within its topmost parent activity.
INVOCATION_ID	INTEGER	An identifier that distinguishes one invocation of a routine from others at the same nesting level within a unit of work. It is unique within a unit of work for a specific nesting level.
ROUTINE_ID	INTEGER	Unique ID of a routine. The value of the column is null if the column is not part of a routine.

Note: The possible combinations of EVENT_STATE, EVENT_TYPE, and EVENT_OBJECT column values are listed in the following table.

Table 333. Possible combinations for EVENT_STATE, EVENT_TYPE, and EVENT_OBJECT column values

Event description	EVENT_STATE value	EVENT_TYPE value	EVENT_OBJECT value
Acquire lock	IDLE	ACQUIRE	LOCK
Escalate lock	EXECUTING	PROCESS	LOCK_ESCALATION
Process request	EXECUTING	PROCESS	REQUEST

Table 333. Possible combinations for EVENT_STATE, EVENT_TYPE, and EVENT_OBJECT column values (continued)

Event description	EVENT_STATE value	EVENT_TYPE value	EVENT_OBJECT value
Wait for a new request	IDLE	WAIT	REQUEST
Wait for a request to be processed at a remote partition	IDLE	WAIT	REMOTE_REQUEST
Wait on a Query Patroller queue	IDLE	WAIT	QP_QUEUE
Wait on a WLM threshold queue	IDLE	WAIT	WLM_QUEUE
Process a routine	EXECUTING	PROCESS	ROUTINE
Recreate an index	EXECUTING	PROCESS	INDEX_RECREATE
Build compression dictionary	EXECUTING	PROCESS	COMP_DICT_BLD
Implicit rebind	EXECUTING	PROCESS	IMPLICIT_REBIND

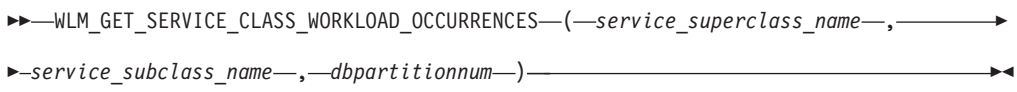
WLM_GET_SERVICE_CLASS_WORKLOAD _OCCURRENCES - List of workload occurrences

Returns the list of all workload occurrences executing in a given service class on a particular partition.

Note: This table function has been deprecated and replaced by the "WLM_GET_SERVICE_CLASS_WORKLOAD _OCCURRENCES_V97 - List workload occurrences" on page 924.

A workload occurrence is a specific database connection whose attributes match with the definition of a workload and hence is associated with or assigned to the workload.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies a valid service superclass name in the currently connected database. If the argument is null or an empty string, the data is retrieved for all the superclasses in the database for which the other parameters match.

service_subclass_name
Target service subclass for the workload occurrence. Any work submitted by

this workload occurrence will run in this service subclass under the target service superclass with the exception of activities that are mapped, or remapped, to a different subclass.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Indicate -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES function.

Example

If an administrator would like to see what workload occurrences are running on the system as a whole, the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES function can be called with a null value or an empty string for *service_superclass_name* and *service_subclass_name*, and -2 for *dbpartitionnum*.

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       SUBSTR(CHAR(COORD_PARTITION_NUM),1,4) AS COORDPART,
       SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHNDL,
       SUBSTR(WORKLOAD_NAME,1,22) AS WORKLOAD_NAME,
       SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES
      (CAST(NULL AS VARCHAR(128)), CAST(NULL AS VARCHAR(128)), -2))
       AS SCINFO
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART, APPHNDL,
       WORKLOAD_NAME, WLO_ID
```

Assuming that the system has four database partitions and is running two workloads at this time, the above query would produce a result like the following:

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART COORDPART ...
------------------- ------------------ ---- --------- ...
SYSDEFAULTMAINTENAN SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTSYSTEMCLA SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    0         ...
```

Output from this query (continued).

```
... APPHNDL WORKLOAD_NAME          WLO_ID
... ------- ---------------------- ------
... -       -                      -
... -       -                      -
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
```

```
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
```

Usage note

The parameters have the effect of being ANDed together. That is, if one were to specify conflicting input parameters such as a service superclass SUP_A and subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows would be returned.

Note: Statistics reported for the workload occurrence (for example coord_act_completed_total) are reset at the beginning of each unit of work when they are combined with the corresponding workload statistics.

Information returned

Table 334. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES

Column Name	Data Type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
COORD_PARTITION_NUM	SMALLINT	Partition number of the coordinator partition of the given workload occurrence.
APPLICATION_HANDLE	BIGINT	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier will be the same on every partition where the application may make a secondary connection.
WORKLOAD_NAME	VARCHAR(128)	Name of the workload from which this record was collected.
WORKLOAD_OCCURRENCE_ID	INTEGER	The ID of the workload occurrence. This does not uniquely identify the workload occurrence unless it is coupled with the coordinator database partition number and the workload name.

Table 334. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES (continued)

Column Name	Data Type	Description
WORKLOAD_OCCURRENCE_STATE	VARCHAR(32)	Possible values include: • `DECOUPLED` - Workload occurrence does not have a coordinator agent assigned (concentrator case). • `DISCONNECTPEND` - Workload occurrence is disconnecting from the database • `FORCED` - Workload occurrence has been forced. • `INTERRUPTED` - Workload occurrence has been interrupted. • `QUEUED` - Workload occurrence coordinator agent is queued by Query Patroller or a workload management queuing threshold. In a partitioned database environment, this state may indicate that the coordinator agent has made an RPC to the catalog partition to obtain threshold tickets and has not yet received a response. • `TRANSIENT` - Workload occurrence has not yet been mapped to a service superclass. • `UOWEXEC` - Workload occurrence is processing a request. • `UOWWAIT` - Workload occurrence is waiting for a request from the client.
UOW_ID	INTEGER	Unique unit of work identifier within an application.
SYSTEM_AUTH_ID	VARCHAR(128)	System authorization ID under which the workload occurrence was injected into the system.
SESSION_AUTH_ID	VARCHAR(128)	Session authorization ID under which the workload occurrence was injected into the system.
APPLICATION_NAME	VARCHAR(128)	The name of the application that created this workload occurrence.
CLIENT_WRKSTNNAME	VARCHAR(255)	The current value of the CLIENT_WRKSTNNAME special register for this workload occurrence.
CLIENT_ACCTNG	VARCHAR(255)	The current value of the CLIENT_ACCTNG special register for this workload occurrence.
CLIENT_USER	VARCHAR(255)	The current value of the CLIENT_USERID special register for this workload occurrence.
CLIENT_APPLNAME	VARCHAR(255)	The current value of the CLIENT_APPLNAME special register for this workload occurrence.
COORD_ACT_COMPLETED_TOTAL	INTEGER	The number of coordinator activities at any nesting level completed so far in the current unit of work of this workload occurrence. This statistic is updated every time an activity in this workload occurrence completes and is reset at the beginning of each unit of work.

Table 334. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES (continued)

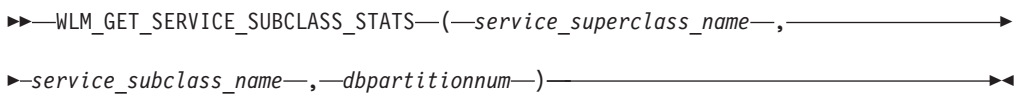
Column Name	Data Type	Description
COORD_ACT_ABORTED_TOTAL	INTEGER	The number of coordinator activities aborted so far in the current unit of work of this workload occurrence. This statistic is updated every time an activity in this workload occurrence is aborted and is reset at the beginning of each unit of work.
COORD_ACT_REJECTED_TOTAL	INTEGER	The number of coordinator activities rejected so far in the current unit of work of this workload occurrence. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action, or a predictive threshold. This statistic is updated every time an activity in this workload occurrence is rejected and is reset at the beginning of each unit of work.
CONCURRENT_ACT_TOP	INTEGER	Highest number of concurrent activities at any nesting level in either executing (which includes idle and waiting) or queued state that has been reached for this workload occurrence in the current unit of work. This statistic is reset at the beginning of each unit of work.

WLM_GET_SERVICE_SUBCLASS_STATS - return statistics of service subclasses

Note: This table function has been deprecated and replaced by the "WLM_GET_SERVICE_SUBCLASS_STATS_V97 table function - Return statistics of service subclasses" on page 928.

This function returns basic statistics of one or more service subclasses.

Syntax



The schema is SYSPROC.

Table function parameters

service_superclass_name
An input argument of type VARCHAR(128) that specifies a valid service superclass name in the same database as the one currently connected to when calling this function. If the argument is null or an empty string, the data is retrieved for all the superclasses in the database.

service_subclass_name
An input argument of type VARCHAR(128) that specifies a valid service subclass name in the same database as the one currently connected to when calling this function. If the argument is null or an empty string, the data is retrieved for all the subclasses in the database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database when calling this function. Specify -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_SUBCLASS_STATS function.

Examples

Example 1: As every activity has to be mapped to a DB2 Service Class prior to being executed, the global state of the system can be regularly monitored using the service class statistics table functions query all the service classes on all the partitions. Note that passing a null value for an argument indicates that the result should not be restricted by that argument, except for the final argument, *dbpartitionnum*, where a value of -2 indicates that data from all database partitions is returned. This example returns service class statistics, such as average activity lifetime and standard deviation in seconds:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
      SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
      SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
      CAST(COORD_ACT_LIFETIME_AVG / 1000 AS DECIMAL(9,3))
        AS AVGLIFETIME,
      CAST(COORD_ACT_LIFETIME_STDDEV / 1000 AS DECIMAL(9,3))
        AS STDDEVLIFETIME,
      SUBSTR(CAST(LAST_RESET AS VARCHAR(30)),1,16) AS LAST_RESET
 FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS(CAST(NULL AS VARCHAR(128)),
      CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
 ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART
```

This is an example of output from this query.

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART ...
------------------- ------------------ ---- ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    ...
```

Output from this query (continued).

```
... AVGLIFETIME STDDEVLIFETIME LAST_RESET
... ----------- -------------- ----------------
...     691.242         34.322 2006-07-24-11.44
...     644.740         22.124 2006-07-24-11.44
...     612.431         43.347 2006-07-24-11.44
...     593.451         28.329 2006-07-24-11.44
```

Example 2: The same table function also gives the highest value for average concurrency of coordinator activities running in the service class for each partition.

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       CONCURRENT_ACT_TOP AS ACTTOP,
       CONCURRENT_WLO_TOP AS CONNTOP
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS(CAST(NULL AS VARCHAR(128)),
       CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART
```

This is an example of output from this query.

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART ACTTOP    CONNTOP
------------------- ------------------ ---- --------- ---------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0           10         7
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1            0         0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2            0         0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3            0         0
```

The output of this table function gives you a good high-level view of the "load" on each partition for a specific database. The output comes from checking the average execution times and numbers of activities. Any significant variations of the high level values returned by these table functions may indicate a change in the load on the system.

Usage notes

Some statistics are returned only if the COLLECT AGGREGATE ACTIVITY DATA and COLLECT AGGREGATE REQUEST DATA settings for the corresponding service subclass are set to a value other than "NONE".

The WLM_GET_SERVICE_SUBCLASS_STATS table function returns one row of data per service subclass and per partition. There is no aggregation across service classes (on a partition) or across partitions (for a service class or more). However, aggregation can be achieved through SQL queries, as shown in the examples above.

The parameters have the effect of being logically united as "AND" clauses. That is, if you specify conflicting input parameters such as a superclass named SUPA and a subclass named SUBB, such that SUBB is not a subclass of SUPA, no rows would be returned.

Information returned

Table 335. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS

Column Name	Data Type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that can occur that will trigger a reset of statistics, which will update this timestamp: • The WLM_COLLECT_STATS procedure is called. • The periodic collection and reset process controlled by the WLM_COLLECT_INT configuration parameter causes a collection and reset. • The database is reactivated. • The service subclass for which statistics are being reported was modified and the change was committed. The LAST_RESET timestamp is in local time.

Table 335. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS (continued)

Column Name	Data Type	Description
COORD_ACT_COMPLETED_TOTAL	BIGINT	The total number of coordinator activities that users have submitted since the last reset and completed successfully. This count is updated as each activity completes. If you remap an activity to a different service subclass, then that activity counts only towards the total of the subclass in which it completes.
COORD_ACT_ABORTED_TOTAL	BIGINT	The total number of coordinator activities that users have submitted since the last reset and completed with errors. This count is updated as each activity aborts. If you remap an activity to a different service subclass, then that activity counts only towards the total of the subclass in which it aborts.
COORD_ACT_REJECTED_TOTAL	BIGINT	The total number of coordinator activities that users have submitted since the last reset and were rejected prior to execution instead of being allowed to execute. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action, or a predictive threshold. This count is updated as each activity gets rejected.
CONCURRENT_ACT_TOP	INTEGER	Highest number of concurrent activities at any nesting level in either executing (which includes idle and waiting) or queued state that has been reached for this service subclass.
COORD_ACT_LIFETIME_TOP	BIGINT	High watermark for coordinator activity lifetime, counted at all nesting levels. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. To use this statistic effectively when the service class includes subclasses that are remapped, you must aggregate the COORD_ACT_LIFETIME_TOP high watermark of the service subclass with that of other subclasses affected by the same remapping threshold or thresholds. You must aggregate these values because an activity can complete after it has been remapped to a different service subclass, but the time the activity spends in other service subclasses before being remapped is counted only towards the service class in which it completes.

Table 335. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS (continued)

Column Name	Data Type	Description
COORD_ACT_LIFETIME_AVG	DOUBLE	Arithmetic mean of lifetime for coordinator activities at nesting level 0 associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. The COORD_ACT_LIFETIME_AVG of a service subclass is unaffected by activities that pass through the subclass but are remapped to a different subclass before they complete.
COORD_ACT_LIFETIME_STDDEV	DOUBLE	Standard deviation of lifetime for coordinator activities at nesting level 0 associated with this service subclass since the last reset. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. This standard deviation is computed from the coordinator activity lifetime histogram and may be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin. The COORD_ACT_LIFETIME_STDDEV of a service subclass is unaffected by activities that pass through the service subclass but are remapped to a different subclass before they complete.
COORD_ACT_EXEC_TIME_AVG	DOUBLE	Arithmetic mean of the execution times for coordinator activities at nesting level 0 associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. The execution time average of a service subclass is unaffected by activities that pass through the subclass but are remapped to a different subclass before they complete.
COORD_ACT_EXEC_TIME_STDDEV	DOUBLE	Standard deviation of the execution times for coordinator activities at nesting level 0 associated with this service subclass since the last reset. Units are milliseconds. This standard deviation is computed from the coordinator activity executetime histogram and might be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin. The execution time standard deviation of a service subclass is unaffected by activities that pass through the subclass but are remapped to a different subclass before they complete.

Table 335. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS (continued)

Column Name	Data Type	Description
COORD_ACT_QUEUE_TIME_AVG	DOUBLE	Arithmetic mean of the queue time for coordinator activities at nesting level 0 associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. The queue time average is counted only towards the service subclass in which the activity was queued.
COORD_ACT_QUEUE_TIME_STDDEV	DOUBLE	Standard deviation of the queue time for coordinator activities at nesting level 0 associated with this service subclass since the last reset. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. This standard deviation is computed from the coordinator activity queuetime histogram and may be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin. The queue time standard deviation is counted only towards the service subclass in which the activity was queued.
NUM_REQUESTS_ACTIVE	BIGINT	The number of requests that are executing in the service subclass at the time this table function is executed.
NUM_REQUESTS_TOTAL	BIGINT	The number of requests to finish executing in this service subclass since the last reset. This applies to any request regardless of its membership in an activity. If COLLECT AGGREGATE REQUEST DATA on this service subclass is set to NONE, the value of this column is NULL. The NUM_REQUESTS_TOTAL of a service subclass is unaffected by requests that pass through the service subclass, but do not complete in it.
REQUEST_EXEC_TIME_AVG	DOUBLE	Arithmetic mean of the execution times for requests associated with this service subclass since the last reset. Units are milliseconds. If the internally tracked average has overflowed, the value -2 is returned. If COLLECT AGGREGATE REQUEST DATA on this service class is set to NONE, the value of this column is NULL. The execution time average of a service subclass is unaffected by requests that pass through the subclass, but do not complete in it.

Table 335. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS (continued)

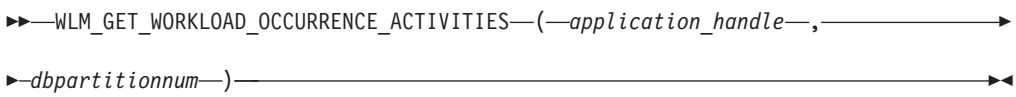
Column Name	Data Type	Description
REQUEST_EXEC_TIME_STDDEV	DOUBLE	Standard deviation of the execution times for requests associated with this service subclass since the last reset. Units are milliseconds. If COLLECT AGGREGATE REQUEST DATA on this service class is set to NONE, the value of this column is NULL. This standard deviation is computed from the request executetime histogram and may be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin. The execution time standard deviation of a service subclass is unaffected by requests that pass through the subclass, but do not complete in it.
REQUEST_EXEC_TIME_TOTAL	BIGINT	Sum of the execution times for requests associated with this service subclass since the last reset. Units are milliseconds. If COLLECT AGGREGATE REQUEST DATA on this service class is set to NONE, the value of this column is NULL. This total is computed from the request execution time histogram and may be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin. The execution time total of a service subclass is unaffected by requests that pass through the subclass, but do not complete in it.

WLM_GET_WORKLOAD_OCCURRENCE _ACTIVITIES - Return a list of activities

The WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES function returns the list of all activities that were submitted by a specified application on a specified partition and have not yet been completed.

Note: This table function has been deprecated and replaced by the "WLM_GET_WORKLOAD_OCCURRENCE _ACTIVITIES_V97 - Return a list of activities" on page 939.

Syntax



The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies an application handle for which a list of activities is to be returned. If the argument is null, the data is retrieved for all the applications in the database.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Specify -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES function.

Example

After you identify the application handle, you can look up all the activities currently running in this application. For example, suppose that an administrator wants to list the activities of an application whose application handle, determined by using the LIST APPLICATIONS command, is 1. The administrator runs the following query:

```
SELECT SUBSTR(CHAR(COORD_PARTITION_NUM),1,5) AS COORD,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       SUBSTR(CHAR(UOW_ID),1,5) AS UOWID,
       SUBSTR(CHAR(ACTIVITY_ID),1,5) AS ACTID,
       SUBSTR(CHAR(PARENT_UOW_ID),1,8) AS PARUOWID,
       SUBSTR(CHAR(PARENT_ACTIVITY_ID),1,8) AS PARACTID,
              ACTIVITY_TYPE AS ACTTYPE,
       SUBSTR(CHAR(NESTING_LEVEL),1,7) AS NESTING
  FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2)) AS WLOACTS
  ORDER BY PART, UOWID, ACTID
```

Sample output from the query is as follows:

```
COORD PART UOWID ACTID PARUOWID PARACTID ACTTYPE  NESTING
----- ---- ----- ----- -------- -------- -------- -------
0     0    2     3     -        -        CALL     0
0     0    2     5     2        3        READ_DML 1
```

Information returned

Table 336. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	System-wide unique ID for the application. On a single-partitioned database, this ID consists of a 16-bit counter. On a multi-partitioned database, this ID consists of the coordinating partition number concatenated with a 16-bit counter. In addition, this ID is the same on every partition where the application makes a secondary connection.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
COORD_PARTITION_NUM	SMALLINT	Coordinator partition of the activity.

Table 336. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES (continued)

Column name	Data type	Description
LOCAL_START_TIME	TIMESTAMP	Local time that this activity began doing work on the partition. The value of the column is null when an activity has entered the system but is in a queue and has not started running.
UOW_ID	INTEGER	Unique ID for the unit of work that the activity started in.
ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work.
PARENT_UOW_ID	INTEGER	Unique ID for the unit of work that the parent activity of the activity started in. If the activity has no parent activity or is at a remote partition, the value of this column is null.
PARENT_ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work for the parent of the activity whose ID is the value of the ACTIVITY_ID column. If the activity has no parent activity or is at a remote partition, the value of this column is null.
ACTIVITY_STATE	VARCHAR(32)	Activity state. Possible values are as follows: **CANCEL_PENDING** The activity was cancelled because there was no agent actively working on a request for the activity. The next time that a request is submitted as part of the activity, the activity will be cancelled, and an SQL4725N error will be generated. **EXECUTING** Agents are actively working on a request for the activity. **IDLE** There is no agent actively processing a request for the activity. **INITIALIZING** The activity has been submitted but has not yet started running. During the initializing state, predictive thresholds are applied to the activity to determine whether the activity will be allowed to run.

Table 336. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES (continued)

Column name	Data type	Description
ACTIVITY_STATE (continued)	VARCHAR(32)	Activity state. Possible values are as follows: **QP_CANCEL_PENDING** This state is the same as the CANCEL_PENDING state except that the activity was cancelled by Query Patroller rather than by the WLM_CANCEL_ACTIVITY procedure. **QP_QUEUED** The activity is queued by Query Patroller. **QUEUED** The activity is queued by a workload management queuing threshold. In a partitioned database environment, this state might mean that the coordinator agent has made an RPC to the catalog partition to obtain threshold tickets and has not yet received a response. This state might indicate that the activity has been queued by a workload management queuing threshold or, if not much time has elapsed, can indicate that the activity is in the process of obtaining its tickets. To obtain a more accurate picture of whether the activity is being queued, determine what agent is working on the activity, and find out whether the value of the EVENT_OBJECT of the agent at the catalog partition is WLM_QUEUE. **TERMINATING** The activity has completed running and is being removed from the system.
ACTIVITY_TYPE	VARCHAR(32)	Activity type. Possible values are as follows: • CALL • DDL • LOAD • OTHER • READ_DML • WRITE_DML Refer to "Identify types of work with work classes" in *Workload Manager Guide and Reference* for a description of the different types of SQL statements that are associated with each activity type.
NESTING_LEVEL	INTEGER	Depth to which this activity is nested within its topmost parent activity.

Table 336. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES (continued)

Column name	Data type	Description
INVOCATION_ID	INTEGER	An identifier that distinguishes one invocation of a routine from others at the same nesting level within a unit of work. It is unique within a unit of work for a specific nesting level.
ROUTINE_ID	INTEGER	Unique ID of the routine.
UTILITY_ID	INTEGER	One of the following values: • If the activity is a utility, the value is the ID of the utility. • If the activity is not a utility, the value is null.
SERVICE_CLASS_ID	INTEGER	Unique ID of the service class to which this activity belongs.
DATABASE_WORK_ACTION_SET_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of database scope, the value is the ID of the work class set of which this work class is a member. • If this activity has not been categorized into a work class of database scope, the value is null.
DATABASE_WORK_CLASS_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of database scope, the value is the ID of the work class. • If this activity has not been categorized into a work class of database scope, the value is null.
SERVICE_CLASS_WORK_ACTION_SET_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of service class scope, the value is the ID of the work action set associated with the work class set to which the work class belongs. • If this activity has not been categorized into a work class of service class scope, the value is null.
SERVICE_CLASS_WORK_CLASS_ID	INTEGER	One of the following values: • If this activity has been categorized into a work class of service class scope, the value is the ID of the work class assigned to this activity. • If this activity has not been categorized into a work class of service class scope, the value is null.

WLM_GET_WORKLOAD_STATS - return workload statistics

Note: This table function has been deprecated and replaced by the "WLM_GET_WORKLOAD_STATS_V97 table function - Return workload statistics" on page 944.

This function returns workload statistics for every combination of workload name and database partition number.

Syntax

The schema is SYSPROC.

Table function parameters

workload_name
An input argument of type VARCHAR(128) that specifies a specific workload for which the statistics are to be returned. If the argument is NULL or an empty string, statistics are returned for all workloads.

dbpartitionnum
An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database when calling this function. Specify -1 for the current database partition, or -2 for all database partitions. If a null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_WORKLOAD_STATS function.

Example

An administrator may want to look at the statistics for workloads. She could do so using the following query:

```
SELECT SUBSTR(WORKLOAD_NAME,1,22) AS WL_DEF_NAME,
     SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
     CONCURRENT_WLO_TOP AS WLO_TOP,
     CONCURRENT_WLO_ACT_TOP AS WLO_ACT_TOP
FROM TABLE(WLM_GET_WORKLOAD_STATS(CAST(NULL AS VARCHAR(128)), -2))
  AS WLSTATS
ORDER BY WL_DEF_NAME, PART
```

The following is an example of output from this query.

```
WL_DEF_NAME            PART WLO_TOP         WLO_ACT_TOP
---------------------- ---- --------------- -------------------
MYUSERWORKLOAD         0                 2                   8
MYUSERWORKLOAD         1                 0                   0
SYSDEFAULTUSERWORKLOAD 0                 1                   1
SYSDEFAULTUSERWORKLOAD 1                 0                   0
```

Here we see that on partition 0, the highest number of concurrent occurrences of the MYUSERWORKLOAD workload was 2 and that the highest number of concurrent activities in either of these workload occurrences was 8.

Usage note

This function returns one row for every combination of workload name and database partition number. No aggregation across workloads or across partitions or across service classes is performed. However, aggregation can be achieved through SQL queries.

Information returned

Table 337. Information returned by WLM_GET_WORKLOAD_STATS

Column Name	Data Type	Description
WORKLOAD_NAME	VARCHAR(128)	Name of the workload from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that can occur that will trigger a reset of statistics, which will update this timestamp: • The WLM_COLLECT_STATS procedure is called. • The periodic collection and reset process controlled by the WLM_COLLECT_INT configuration parameter causes a collection and reset. • The database is reactivated. • The workload for which statistics are being reported was modified and the change was committed. The LAST_RESET timestamp is in local time.
CONCURRENT_WLO_TOP	INTEGER	Highest number of concurrent occurrences of the given workload on this partition since the last reset.
CONCURRENT_WLO_ACT_TOP	INTEGER	Highest number of concurrent activities (including both coordinator and nested) in either executing (which includes idle and waiting) or queued state that has been reached in any occurrence of this workload since last reset. Updated by each workload occurrence at the end of its unit of work.
COORD_ACT_COMPLETED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level assigned to any occurrence of this workload that completed since the last reset. Updated by each workload occurrence at the end of its unit of work.
COORD_ACT_ABORTED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level assigned to any occurrence of this workload that were aborted prior to completion since the last reset. Updated by each workload occurrence at the end of its unit of work.

Table 337. Information returned by WLM_GET_WORKLOAD_STATS (continued)

Column Name	Data Type	Description
COORD_ACT_REJECTED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level assigned to any occurrence of this workload that were rejected prior to execution since the last reset. Updated by each workload occurrence at the end of its unit of work. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action, or a predictive threshold. Note that unlike the column of the same name in the WLM_GET_SERVICE_SUBCLASS_STATS function, this also counts rejections that occur before an activity can be assigned to a service class. An example of such a rejection occurs when an activity violates the ConcurrentWorkloadOccurrences threshold.
WLO_COMPLETED_TOTAL	BIGINT	The number of workload occurrences to complete since last reset.

Appendix A. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an e-mail to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss. English and translated DB2 Version 9.7 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 338. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC27-2435-02	Yes	September, 2010
Administrative Routines and Views	SC27-2436-02	No	September, 2010
Call Level Interface Guide and Reference, Volume 1	SC27-2437-02	Yes	September, 2010
Call Level Interface Guide and Reference, Volume 2	SC27-2438-02	Yes	September, 2010
Command Reference	SC27-2439-02	Yes	September, 2010
Data Movement Utilities Guide and Reference	SC27-2440-00	Yes	August, 2009
Data Recovery and High Availability Guide and Reference	SC27-2441-02	Yes	September, 2010
Database Administration Concepts and Configuration Reference	SC27-2442-02	Yes	September, 2010
Database Monitoring Guide and Reference	SC27-2458-02	Yes	September, 2010
Database Security Guide	SC27-2443-01	Yes	November, 2009
DB2 Text Search Guide	SC27-2459-02	Yes	September, 2010
Developing ADO.NET and OLE DB Applications	SC27-2444-01	Yes	November, 2009
Developing Embedded SQL Applications	SC27-2445-01	Yes	November, 2009
Developing Java Applications	SC27-2446-02	Yes	September, 2010
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-2447-01	No	September, 2010
Developing User-defined Routines (SQL and External)	SC27-2448-01	Yes	November, 2009
Getting Started with Database Application Development	GI11-9410-01	Yes	November, 2009
Getting Started with DB2 Installation and Administration on Linux and Windows	GI11-9411-00	Yes	August, 2009

Table 338. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Globalization Guide	SC27-2449-00	Yes	August, 2009
Installing DB2 Servers	GC27-2455-02	Yes	September, 2010
Installing IBM Data Server Clients	GC27-2454-01	No	September, 2010
Message Reference Volume 1	SC27-2450-00	No	August, 2009
Message Reference Volume 2	SC27-2451-00	No	August, 2009
Net Search Extender Administration and User's Guide	SC27-2469-02	No	September, 2010
Partitioning and Clustering Guide	SC27-2453-01	Yes	November, 2009
pureXML Guide	SC27-2465-01	Yes	November, 2009
Query Patroller Administration and User's Guide	SC27-2467-00	No	August, 2009
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC27-2468-01	No	September, 2010
SQL Procedural Languages: Application Enablement and Support	SC27-2470-02	Yes	September, 2010
SQL Reference, Volume 1	SC27-2456-02	Yes	September, 2010
SQL Reference, Volume 2	SC27-2457-02	Yes	September, 2010
Troubleshooting and Tuning Database Performance	SC27-2461-02	Yes	September, 2010
Upgrading to DB2 Version 9.7	SC27-2452-02	Yes	September, 2010
Visual Explain Tutorial	SC27-2462-00	No	August, 2009
What's New for DB2 Version 9.7	SC27-2463-02	Yes	September, 2010
Workload Manager Guide and Reference	SC27-2464-02	Yes	September, 2010
XQuery Reference	SC27-2466-01	No	November, 2009

Table 339. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
Installing and Configuring DB2 Connect Personal Edition	SC27-2432-02	Yes	September, 2010
Installing and Configuring DB2 Connect Servers	SC27-2433-02	Yes	September, 2010

Table 339. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print	Last updated
DB2 Connect User's Guide	SC27-2434-02	Yes	September, 2010

Table 340. Information Integration technical information

Name	Form Number	Available in print	Last updated
Information Integration: Administration Guide for Federated Systems	SC19-1020-02	Yes	August, 2009
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-04	Yes	August, 2009
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-02	No	August, 2009
Information Integration: SQL Replication Guide and Reference	SC19-1030-02	Yes	August, 2009
Information Integration: Introduction to Replication and Event Publishing	GC19-1028-02	Yes	August, 2009

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* DVD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation DVD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation DVD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v9r7.

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:

1. Locate the contact information for your local representative from one of the following Web sites:
   - The IBM directory of world wide contacts at www.ibm.com/planetwide
   - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
2. When you call, specify that you want to order a DB2 publication.
3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 1223.

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.
For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

- To display topics in your preferred language in the Internet Explorer browser:
  1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.

2. Ensure your preferred language is specified as the first entry in the list of languages.
   – To add a new language to the list, click the **Add...** button.

     Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
   – To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Refresh the page to display the DB2 Information Center in your preferred language.

- To display topics in your preferred language in a Firefox or Mozilla browser:
  1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     – To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
     – To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you must also change the regional settings of your operating system to the locale and language of your choice.

Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

A DB2 Version 9.7 Information Center must already be installed. For details, see the "Installing the DB2 Information Center using the DB2 Setup wizard" topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates - updates existing Information Center features and languages. An additional benefit of automatic updates is that the Information Center is unavailable for a minimal period of time during the update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates - should be used when you want to add features or languages during the update process. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process.

This topic details the process for automatic updates. For manual update instructions, see the "Manually updating the DB2 Information Center installed on your computer or intranet server" topic.

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
   a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9.7` directory.
   b. Navigate from the installation directory to the `doc/bin` directory.
   c. Run the `ic-update` script:

      ```
      ic-update
      ```
2. On Windows operating systems,
   a. Open a command window.
   b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `<Program Files>\IBM\DB2 Information Center\Version 9.7` directory, where `<Program Files>` represents the location of the Program Files directory.
   c. Navigate from the installation directory to the `doc\bin` directory.
   d. Run the `ic-update.bat` file:

      ```
      ic-update.bat
      ```

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in `doc\eclipse\configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

Updating your locally-installed *DB2 Information Center* manually requires that you:

1. Stop the *DB2 Information Center* on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

   Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.

If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.

3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

Note: On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv97 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the *Program_Files*`\IBM\DB2 Information Center\Version 9.7` directory, where *Program_Files* represents the location of the `Program Files` directory.
     c. Navigate from the installation directory to the `doc\bin` directory.
     d. Run the `help_start.bat` file:
        ```
        help_start.bat
        ```
   - On Linux:
     a. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the `/opt/ibm/db2ic/V9.7` directory.
     b. Navigate from the installation directory to the `doc/bin` directory.
     c. Run the `help_start` script:
        ```
        help_start
        ```

   The systems default Web browser opens to display the stand-alone Information Center.
3. Click the **Update** button (). (JavaScript™ must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
   - On Windows, navigate to the installation directory's `doc\bin` directory, and run the `help_end.bat` file:
     ```
     help_end.bat
     ```

     Note: The `help_end` batch file contains the commands required to safely stop the processes that were started with the `help_start` batch file. Do not use `Ctrl-C` or any other method to stop `help_start.bat`.

- On Linux, navigate to the installation directory's doc/bin directory, and run the help_end script:

  ```
  help_end
  ```

  Note: The help_end script contains the commands required to safely stop the processes that were started with the help_start script. Do not use any other method to stop the help_start script.

7. Restart the *DB2 Information Center*.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv97 start
     ```

The updated *DB2 Information Center* displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click the title.

"pureXML" in *pureXML Guide*
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

"Visual Explain" in *Visual Explain Tutorial*
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation
Troubleshooting information can be found in the *Troubleshooting and Tuning Database Performance* or the Database fundamentals section of the *DB2 Information Center*. There you will find information about how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 database products.

DB2 Technical Support Web site
Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes,

Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/support/db2_9/

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel®, Intel logo, Intel Inside®, Intel Inside logo, Intel® Centrino®, Intel Centrino logo, Celeron®, Intel® Xeon®, Intel SpeedStep®, Itanium®, and Pentium® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft®, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Printed in USA

SC27-2436-02

Spine information:

IBM DB2 9.7 for Linux, UNIX, and Windows **Version 9 Release 7** **Administrative Routines and Views**