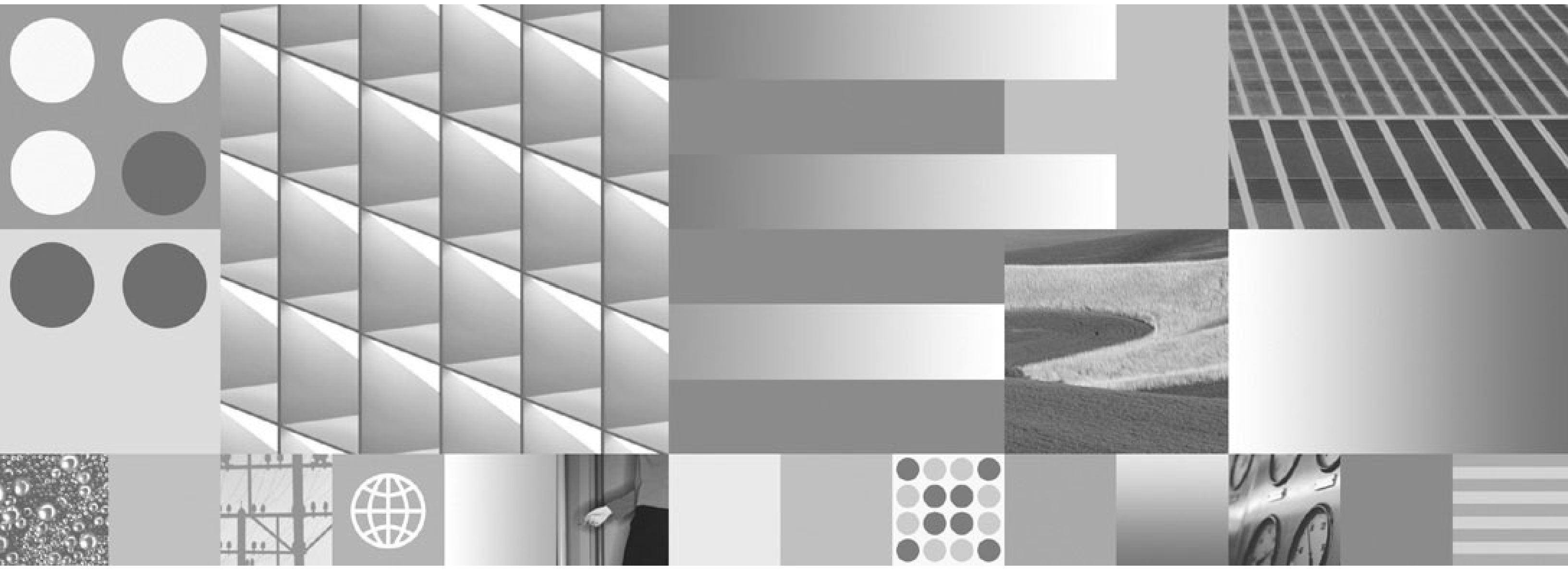
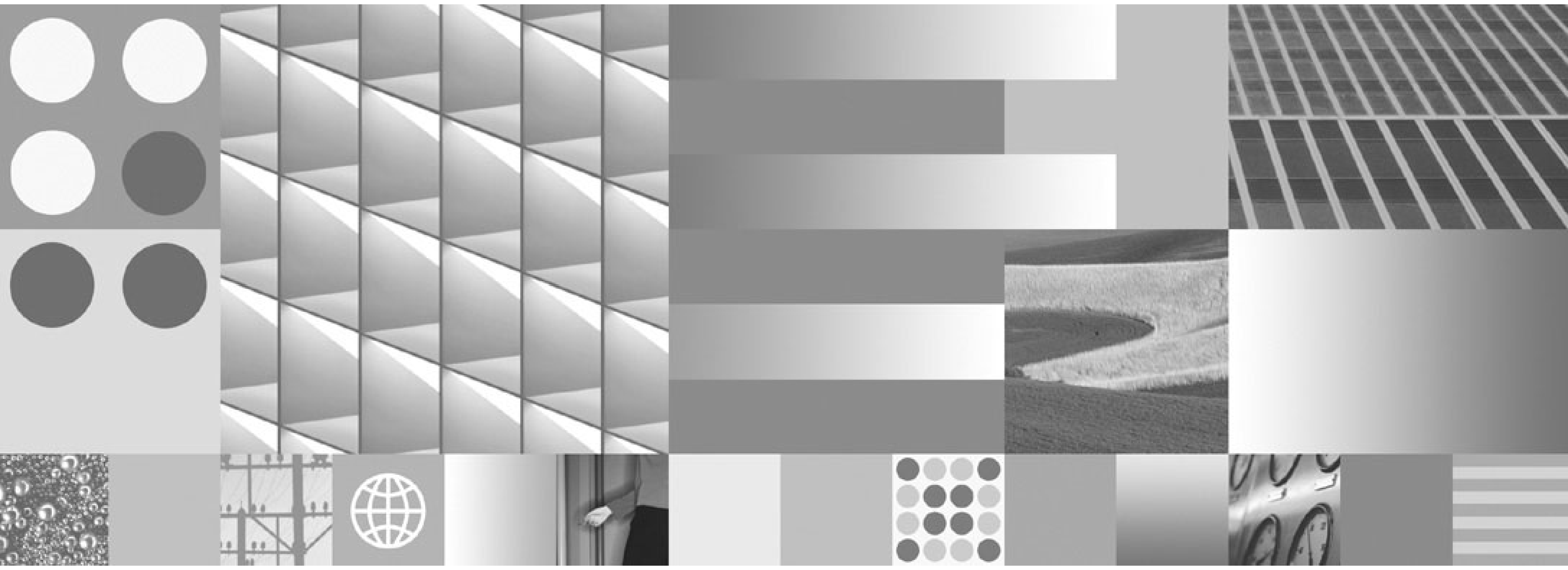
IBM DB2 9.7
for Linux, UNIX, and Windows



Data Recovery and High Availability Guide and Reference

SC27-2441-00

IBM DB2 9.7
for Linux, UNIX, and Windows



Data Recovery and High Availability Guide and Reference

SC27-2441-00

Note

Before using this information and the product it supports, read the general information under Appendix B, “Notices,” on page 351.

Contents

About this book

The Data Recovery and High Availability Guide and Reference describes how to keep your DB2® Database for Linux®, UNIX®, and Windows® database solutions highly available, and how to keep your data from being lost.

The Data Recovery and High Availability Guide and Reference is in two parts:

- Part 1, High availability, describes strategies and DB2 database features and functionality that help keep your database solutions highly available.
- Part 2, Data recovery, describes how to use DB2 backup and restore functionality to keep your data from being lost.

Part 1. High availability

The availability of a database solution is a measure of how successful user applications are at performing their required database tasks. If user applications cannot connect to the database, or if their transactions fail because of errors or time out because of load on the system, the database solution is not very available. If user applications are successfully connecting to the database and performing their work, the database solution is highly available.

Designing a highly available database solution, or increasing the availability of an existing solution requires an understanding of the needs of the applications accessing the database. To get the greatest benefit from the expense of additional storage space, faster processors, or more software licenses, focus on making your database solution as available as required to the most important applications for your business at the time when those applications need it most.

Unplanned outages

Unexpected system failures that could affect the availability of your database solution to users include: power interruption; network outage; hardware failure; operating system or other software errors; and complete system failure in the event of a disaster. If such a failure occurs at a time when users expect to be able to do work with the database, a highly available database solution must do the following:

- Shield user applications from the failure, so the user applications are not aware of the failure. For example, DB2 Data Server can reroute database client connections to alternate database servers if a database server fails.
- Respond to the failure to contain its effect. For example, if a failure occurs on one machine in a cluster, the cluster manager can remove that machine from the cluster so that no further transactions are routed to be processed on the failed machine.
- Recover from the failure to return the system to normal operations. For example, if standby database takes over database operations for a failed primary database, the failed database might restart, recover, and take over once again as the primary database.

These three tasks must be accomplished with a minimum effect on the availability of the solution to user applications.

Planned outage

In a highly available database solution, the impact of maintenance activities on the availability of the database to user applications must be minimized as well.

For example, if the database solution serves a traditional store front that is open for business between the hours of 9am to 5pm, then maintenance activities can occur offline, outside of those business hours without affecting the availability of the database for user applications. If the database solution serves an online banking business that is expected to be available for customers to access through the Internet 24 hours per day, then maintenance activities must be run online, or scheduled for off-peak activity periods to have minimal impact on the availability of the database to the customers.

When you are making business decisions and design choices about the availability of your database solution, you must weigh the following two factors:

- The cost to your business of the database being unavailable to customers
- The cost of implementing a certain degree of availability

For example, consider an Internet-based business that makes a certain amount of revenue, X, every hour the database solution is serving customers. A high availability strategy that saves 10 hours of downtime per year will earn the business 10X extra revenue per year. If the cost of implementing this high availability strategy is less than the expected extra revenue, it would be worth implementing.

Chapter 1. Outages

An outage is any disruption in the ability of the database solution to serve user applications. Outages can be classified in two groups: unplanned outages and planned outages.

Unplanned outages

Examples of unplanned outages include:

- The failure of one component of the system, including hardware or software failure.
- Invalid administrative or user application actions such accidentally dropping a table that is needed for business-critical transactions.
- Poor performance due to suboptimal configuration, or inadequate hardware or software.

Planned outages

Examples of planned outages include:

- Maintenance. Some maintenance activities require you to take a complete outage; other maintenance activities can be performed without stopping the database, but can adversely affect performance. The latter is the most common type of planned outage.
- Upgrade. Upgrading your software or hardware can sometimes require a partial or a full outage.

In discussions about availability, the focus is often on disaster scenarios or component failures. However, to design a robust high availability solution, you need to address all of these types of outage.

Outage signatures

An outage signature is a collection of symptoms and behaviors which characterize an outage. The signature of an outage may vary from temporary performance issues resulting in slow response time for end users to complete site failure. Consider how these variations impact your business when devising strategies for avoiding, minimizing, and recovering from outages.

Blackout

A blackout type of outage is experienced when a system is completely unavailable to its end users. This type of outage may be caused by problems at the hardware, operating system, or database level. When a blackout occurs, it is imperative that the scope of the outage is immediately identified. Is the outage purely at the database level? Is the outage at the instance level? Or is it at the operating system or hardware level?

Brownout

A brownout type of outage is experienced when system performance slows to a point where end users cannot effectively get their work done. The system as a whole may be up and running, but essentially, in the eyes of the end users it is not working as expected. This type of outage may occur during system maintenance windows and peak usage periods. Typically, the CPU and memory are near capacity during such outages. Poorly tuned or overutilized servers often contribute to brownouts.

Frequency and duration of outages

In conversations about database availability, the focus is often on the total amount or the percentage of down time (or conversely the amount of time the database system is available) for a given time period. However, the frequency and duration of planned or unplanned outages makes a significant difference to the impact that those outages have on your business.

Consider a situation in which you have to make some upgrades to your database system that will take seven hours to perform, and you can choose between taking the database system offline for an hour every day during a period of low user activity or taking the database offline for seven hours during the busiest part of your busiest day. Clearly, several small outages would be less costly and harmful to your business activities than the single, seven-hour outage. Now consider a situation in which you have intermittent network failures, possibly for a total of a few minutes every week, which cause a small number of transactions to fail with regular frequency. Those very short outages might end up costing you a great deal of revenue, and irreparably damage the confidence of your customers in your business–resulting in even greater losses of future revenue.

Don't focus exclusively on the total outage (or available) time. Weigh the cost of fewer, longer outages against the cost of multiple, smaller outages when making decisions about maintenance activities or when responding to an unplanned outage. In the middle of an outage, it can be difficult to make such judgments; so create a formula or method to calculate the cost to your business of these outage signatures so that you can make the best choices.

Multiple and cascading failures

When you are designing your database solution to avoid, minimize, and recover from outages, keep in mind the possibility for multiple components to fail at the same time, or even for the failure of one component to cause another component to fail.

Outage cost

The cost of an outage varies from business to business. Each business, as a best practice, should analyze the cost of an outage to their mission critical business processes. The results of this analysis are used to formulate a restoration plan. This plan includes a priority ordering among restoration activities if more than one process is identified.

Outage cost

You can estimate the cost to your business of your customer-facing database system being unavailable to process customer transactions. For example, you can calculate an average cost in lost sales revenue for every hour or minute during which that database system is unavailable. Calculating projected losses in revenue as a result of reduced customer confidence is much more difficult, but you should consider this cost when assessing your business's availability requirements.

Consider too the cost of internal database systems being unavailable to your business processes. Something as simple as email or calendar software being unavailable for an hour can cause your business to grind a halt, because employees are unable to do their work.

Outage tolerance

The tolerance of an outage varies from business to business. Each business, as a best-practice, should analyze the impact of an outage to their mission critical business processes. The results of this analysis are used to formulate a restoration plan. This plan includes an order of priority to the restoration if more than one process is identified.

Outage tolerance

A crucial factor in determining your availability needs is to ask how tolerant your business, or a specific system in your business, is to the occurrence of an outage. For example, a restaurant that operates a website primarily to publish menu information will not lose much revenue because of an occasional server outage. On the other hand, any outage on a stock exchange server that records transactions would be catastrophic. Thus, using a lot of resources to ensure the availability of the restaurant's server is 99.99% would not be cost-effective, whereas it certainly would be for the stock exchange.

When discussing tolerance two concepts should be kept in mind: time to recovery, and point of recovery.

Time to recovery is the time required to bring a business process or system back online.

Point of recovery is the historical point at which the business process or system is restored. In database terms, a plan would weigh the benefits of a quick restore that loses some transactions versus a complete restore that loses no transactions but which takes longer to perform.

Recovery and avoidance strategies

When considering purchase and system design choices about availability, it is tempting to dive into long lists of high availability features and technologies. However, best practices with respect to making and keeping your system highly available are just as much about making good design and configuration choices, and designing and practicing sound administrative procedures and emergency plans, as they are about buying technology.

You will get the most comprehensive availability for your investment by first identifying the high availability strategies that best suit your business demands. Then you can implement your strategies, choosing the most appropriate technology.

When designing or configuring your database solution for high availability, consider how outages may be avoided, their impact minimized, and your system quickly recovered.

Avoid outages

Whenever possible, avoid outages. For example, remove single points of failure to avoid unplanned outages, or investigate methods for performing maintenance activities online to avoid planned outages. Monitor your database system to identify trends in system behavior that indicate problems, and resolve the problems before they cause an outage.

Minimize the impact of outages

You can design and configure your database solution to minimize the impact of planned and unplanned outages. For example, distribute your database solution so that components and functionality are localized, allowing some user applications to continue processing transactions even when one component is offline.

Recover quickly from unplanned outages

Make a recovery plan: create clear and well-documented procedures that administrators can follow easily and quickly in the event of an unplanned outage; create clear architectural documents that describe all components of the systems involved; have service agreements and contact information well organized and close to hand. While recovering quickly is vitally important, also know what diagnostic information to collect in order to identify the root cause of the outage and avoid it in the future.

Chapter 2. High availability strategies

It does not matter to a user why his or her database request failed. Whether a transaction timed out because of bad performance, or a component of the solution failed, or an administrator has taken the database offline to perform maintenance, the result is the same to the user: the database is unavailable to process requests.

Strategies for improving the availability of your database solution include:

Redundancy
: having secondary copies of each component of your solution that can take over workload in the event of failure.

System monitoring
: collecting statistics about the components of your solution to facilitate workload balancing or detecting that components have failed.

Load balancing
: transferring some workload from an overloaded component of your solution to another component of your solution that has a lighter load.

Failover
: transferring all workload from a failed component of your solution to a secondary component.

Maximizing performance
: reducing the chance that transactions take a very long time to complete or time out.

Minimizing the impact of maintenance
: scheduling automated maintenance activities and manual maintenance activities so as to impact user applications as little as possible.

High availability through redundancy

If any one element or component of your database solution fails, from the power supply or cable connecting your network, to the operating system or application software, the end result is the same: your database solution is not available to user applications. An important strategy for maintaining high availability is having redundant systems so that if one component fails, a secondary or backup copy of that component can take over for the failed component, enabling the database to remain available to user applications.

Redundancy is common in system design:
- Uninterrupted or backup power supplies
- Running multiple network fibers between each component
- Bonding or load balancing of network cards
- Using multiple hard drives in a redundant array
- Using clusters of CPUs

If any one of these components of the system is not redundant, that component could be a single point of failure for the whole system.

You can create redundancy at the database level, by having two databases: a primary database that normally processes all or most of the application workload;

and a secondary database that can take over the workload if the primary database fails. In a DB2 High Availability Disaster Recover (HADR) environment, this secondary database is called the standby database.

High availability through failover

Failover is the transfer of workload from a primary system to a secondary system in the event of a failure on the primary system. When workload has been transferred like this, the secondary system is said to have taken over the workload of the failed primary system.

Example 1

In a clustered environment, if one machine in the cluster fails, cluster managing software can move processes that were running on the machine that failed to another machine in the cluster.

Example 2

In a database solution with multiple IBM® Data Servers, if one database becomes unavailable, the database manager can reroute database applications that were connected to the database server that is no longer available to a secondary database server.

The two most common failover strategies on the market are known as idle standby and mutual takeover:

Idle Standby

In this configuration, a primary system processes all the workload while a secondary or standby system is idle, or in standby mode, ready to take over the workload if there is a failure on the primary system. In an HADR setup, the standby can also be configured to allow read-only workloads.

Mutual Takeover

In this configuration, there are multiple systems, and each system is the designated secondary for another system. When a system fails, the overall performance is negatively affected because the secondary for the system that failed must continue to process its own workload as well as the workload of the failed system.

High availability through clustering

A cluster is a group of connected machines that work together as a single system. When one machine in a cluster fails, cluster managing software transfers the workload of the failed machine onto other machines.

Heartbeat monitoring

To detect a failure on one machine in the cluster, failover software can use heartbeat monitoring or keepalive packets between machines to confirm availability. Heartbeat monitoring involves system services that maintain constant communication between all the machines in a cluster. If a heartbeat is not detected, failover to a backup machine starts.

IP address takeover

When there is a failure on one machine in the cluster, cluster managers can transfer workload from one machine to another by transferring the IP address from one machine to another. This is called IP address takeover, or

IP takeover. This transfer is invisible to client applications, which continue to use the original IP address, unaware that the physical machine to which that IP address maps has changed.

The DB2 High Availability (HA) Feature enables integration between IBM Data Server and cluster managing software.

Database logging

All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure. Database logging is an important part of your highly available database solution design because database logs make it possible to recover from a failure, and they make it possible to synchronize primary and secondary databases.

IBM Data Server supports two types of logging: *circular* and *archive*. Each provides a different level of recovery capability:
- “Circular logging”
- “Archive logging” on page 10

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to restore a database either to the end of the logs, or to a specific point in time. The archived log files can be used to recover changes made after the backup was taken. This is different from circular logging where you can only recover to the time of the backup, and all changes made after that are lost.

Circular logging

Circular logging is the default behavior when a new database is created. (The *logarchmeth1* and *logarchmeth2* database configuration parameters are set to `OFF`.) With this type of logging, only full, offline backups of the database are allowed. The database must be offline (inaccessible to users) when a full backup is taken.

As the name suggests, circular logging uses a “ring” of online logs to provide recovery from transaction failures and system crashes. The logs are used and retained only to the point of ensuring the integrity of current transactions. Circular logging does not allow you to roll a database forward through transactions performed after the last full backup operation. All changes occurring since the last backup operation are lost. Since this type of restore operation recovers your data to the specific point in time at which a full backup was taken, it is called *version recovery*.

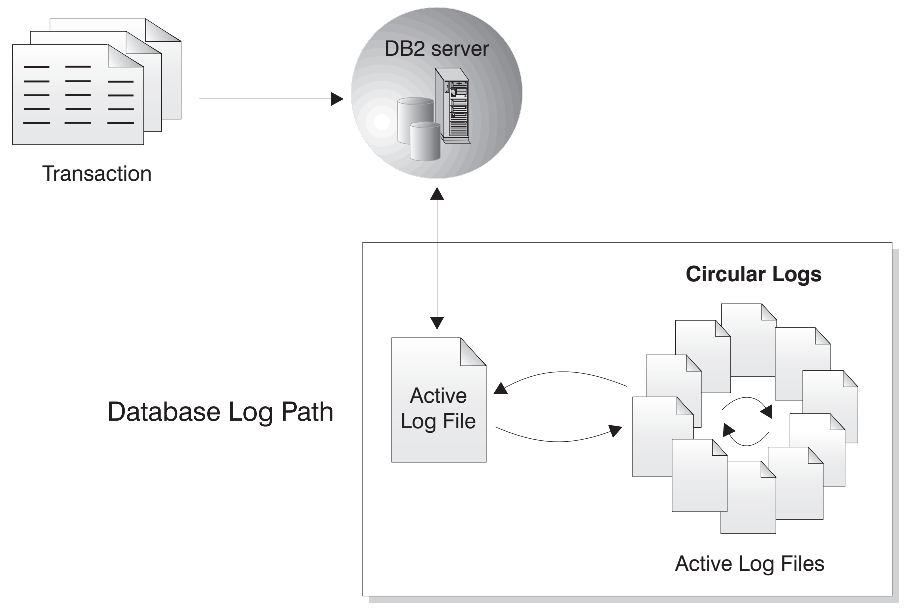
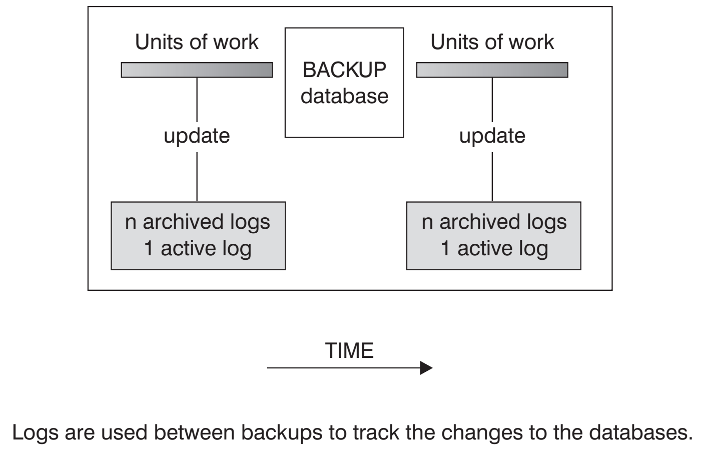


Figure 1. Circular Logging

Active logs are used during crash recovery to prevent a failure (system power or application error) from leaving a database in an inconsistent state. Active logs are located in the database log path directory.

Archive logging

Archive logging is used specifically for rollforward recovery. Archived logs are logs that were active but are no longer required for crash recovery. Use the *logarchmeth1* database configuration parameter to enable archive logging.



Logs are used between backups to track the changes to the databases.

Figure 2. Active and Archived Database Logs in Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

Taking online backups is only supported if the database is configured for archive logging. During an online backup operation, all activities against the database are logged. When an online backup image is restored, the logs must be rolled forward at least to the point in time at which the backup operation completed. For this to happen, the logs must have been archived and made available when the database is restored. After an online backup is complete, the DB2 database manager forces the currently active log to be closed, and as a result, it will be archived. This ensures that your online backup has a complete set of archived logs available for recovery.

The following database configuration parameters allow you to change where archived logs are stored: The *newlogpath* parameter, and the *logarchmeth1* and *logarchmeth2* parameters. Changing the *newlogpath* parameter also affects where active logs are stored.

To determine which log *extents* in the database log path directory are archived logs, check the value of the *loghead* database configuration parameter. This parameter indicates the lowest numbered log that is active. Those logs with sequence numbers less than *loghead* are archived logs and can be moved. You can check the value of this parameter by using the Control Center; or, by using the command line processor and the GET DATABASE CONFIGURATION command to view the "First active log file". For more information about this configuration parameter, see the *Performance Guide* book.

Log control files

When a database restarts after a failure, the database manager applies transaction information stored in log files to return the database to a consistent state. To determine which records from the log files need to be applied to the database, the database manager uses information recorded in a log control file.

Redundancy for database resilience

The database manager maintains two copies of the log control file, SQLOGCTL.LFH.1 and SQLOGCTL.LFH.2, so that if one copy is damaged, the database manager can still use the other copy.

Performance considerations

Applying the transaction information contained in the log control files contributes to the overhead of restarting a database after a failure. You can configure the frequency at which the database manager writes transaction to disk in order to reduce the number of log records that need to be processed during crash recovery using the “softmax - Recovery range and soft checkpoint interval configuration parameter” in *Data Servers, Databases, and Database Objects Guide*.

Chapter 3. High availability with IBM Data Server

IBM Data Server contains functionality that supports many high availability strategies.

Automatic client reroute roadmap

Automatic client reroute is an IBM Data Server feature that redirects client applications from a failed server to an alternate server so the applications can continue their work with minimal interruption. Automatic client reroute can be accomplished only if an alternate server has been specified prior to the failure.

Table 1 lists the relevant topics in each category.

Table 1. Roadmap to automatic client reroute information

Category	Related topics
General information	• "Automatic client reroute limitations" on page 26 • "Automatic client reroute description and setup" on page 21 • "Automatic client reroute description and setup (DB2® Connect™)" in *Quick Beginnings for DB2 Connect Servers*
Configuration	• "Identifying an alternate server for automatic client reroute" on page 25 • "Configuring automatic client reroute retry behavior using registry variables" on page 23 • "Client reroute setup when using IBM Data Server Driver for JDBC and SQLJ" on page 25
Examples	• "Automatic client reroute examples" on page 163
Interaction with other DB2 features	• "Configuring automatic client reroute and High Availability Disaster Recovery (HADR)" on page 33 • "Using client connection timeout with automatic client reroute" on page 24 • "IBM Data Server Driver for JDBC and SQLJ client reroute support" in *Developing Java Applications*
Troubleshooting	• "Configuring automatic client reroute for client connection distributor technology" on page 24

Note: Automatic client reroute for DB2 for z/OS® Sysplex is also available in IBM data server clients and non-Java IBM data server drivers. With this support, applications that access a DB2 for z/OS Sysplex can use automatic client reroute capabilities provided by the client, and are not required to go through a DB2 Connect server. For more information about this feature, see the topic about automatic client reroute (client-side) in the DB2 Information Center.

DB2 fault monitor facilities for Linux and UNIX

Available on UNIX based systems only, DB2 fault monitor facilities keep DB2 Data Server databases up and running by monitoring DB2 database manager instances, and restarting any instance that exits prematurely.

The Fault Monitor Coordinator (FMC) is the process of the Fault Monitor Facility that is started at the UNIX boot sequence. The *init* daemon starts the FMC and will restart it if it terminates abnormally. The FMC starts one fault monitor for each DB2 instance. Each fault monitor runs as a daemon process and has the same user privileges as the DB2 instance.

Once a fault monitor is started, it will be monitored to make sure it does not exit prematurely. If a fault monitor fails, it will be restarted by the FMC. Each fault monitor will, in turn, be responsible for monitoring one DB2 instance. If the DB2 instance exits prematurely, the fault monitor will restart it. The fault monitor will only become inactive if the db2stop command is issued. If a DB2 instance is shut down in any other way, the fault monitor will start it up again.

DB2 fault monitor restrictions

If you are using a high availability clustering product such as HACMP™, MSCS, or IBM Tivoli® System Automation for Multiplatforms, the fault monitor facility must be turned off since the instance startup and shut down is controlled by the clustering product.

Differences between the DB2 fault monitor and the DB2 health monitor

The health monitor and the fault monitor are tools that work on a single database instance. The health monitor uses *health indicators* to evaluate the health of specific aspects of database manager performance or database performance. A health indicator measures the health of some aspect of a specific class of database objects, such as a table space. Health indicators can be evaluated against specific criteria to determine the health of that class of database object. In addition, health indicators can generate alerts to notify you when an indicator exceeds a threshold or indicates a database object is in a non-normal state.

By comparison, the fault monitor is solely responsible for keeping the instance it is monitoring up and running. If the DB2 instance it is monitoring terminates unexpectedly, the fault monitor restarts the instance. The fault monitor is not available on Windows.

High Availability Disaster Recovery (HADR)

The DB2 Data Server High Availability Disaster Recovery (HADR) feature is a database replication feature that provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby.

HADR might be your best option if most or all of your database requires protection, or if you perform DDL operations that must be automatically replicated on the standby database.

Applications can only access the current primary database. Updates to the standby database occur by rolling forward log data that is generated on the primary database and shipped to the standby database.

A partial site failure can be caused by a hardware, network, or software (DB2 database system or operating system) failure. Without HADR, a partial site failure requires restarting the database management system (DBMS) server that contains the database. The length of time it takes to restart the database and the server where it resides is unpredictable. It can take several minutes before the database is brought back to a consistent state and made available. With HADR, the standby database can take over in seconds. Further, you can redirect the clients that were using the original primary database to the standby database (new primary database) by using automatic client reroute or retry logic in the application.

A complete site failure can occur when a disaster, such as a fire, causes the entire site to be destroyed. Because HADR uses TCP/IP for communication between the primary and standby databases, they can be situated in different locations. For example, your primary database might be located at your head office in one city, while your standby database is located at your sales office in another city. If a disaster occurs at the primary site, data availability is maintained by having the remote standby database take over as the primary database with full DB2 functionality. After a takeover operation occurs, you can bring the original primary database back up and return it to its primary database status; this is known as failback.

With HADR, you can choose the level of protection you want from potential loss of data by specifying one of three synchronization modes: synchronous, near synchronous, or asynchronous.

After the failed original primary server is repaired, it can rejoin the HADR pair as a standby database if the two copies of the database can be made consistent. After the original primary database is reintegrated into the HADR pair as the standby database, you can switch the roles of the databases to enable the original primary database to once again be the primary database.

HADR is only one of several replication solutions offered in the DB2 product family. WebSphere® Federation Server and the DB2 database system include SQL replication and Q replication solutions that can also be used, in some configurations, to provide high availability. These functions maintain logically consistent copies of database tables at multiple locations. In addition, they provide flexibility and complex functionality such as support for column and row filtering, data transformation, updates to any copy of a table, and they can be used in partitioned database environments.

DB2 High Availability (HA) Feature

The DB2 High Availability (HA) Feature enables integration between IBM Data Server and cluster managing software.

When you stop a database manager instance in a clustered environment, you must make your cluster manager aware that the instance is stopped. If the cluster manager is not aware that the instance is stopped, the cluster manager might attempt an operation such as failover on the stopped instance. The DB2 High Availability (HA) Feature provides infrastructure for enabling the database manager to communicate with your cluster manager when instance configuration changes, such as stopping a database manager instance, require cluster changes.

If the database manager communicates with the cluster manager whenever instance changes require cluster changes, then you are freed from having to perform separate cluster operations after performing instance configuration changes.

The DB2 HA Feature is composed of the following elements:

- IBM Tivoli System Automation for Multiplatforms (SA MP) is bundled with IBM Data Server on AIX® and Linux as part of the DB2 High Availability (HA) Feature, and integrated with the DB2 installer. You can install, upgrade, or uninstall SA MP using either the DB2 installer or the installSAM and uninstallSAM scripts that are included in the IBM Data Server install media.
- In a clustered environment, some database manager instance configuration and administration operations require related cluster configuration changes. The DB2 High Availability (HA) Feature enables the database manager to automatically request cluster manager configuration changes whenever you perform certain database manager instance configuration and administration operations. See: "Configuring a cluster automatically with the DB2 High Availability (HA) Feature" on page 70
- DB2 High Availability Instance Configuration Utility (db2haicu) is a text based utility that you can use to configure and administer your highly available databases in a clustered environment. db2haicu collects information about your database instance, your cluster environment, and your cluster manager by querying your system. You supply more information through parameters to the db2haicu call, an input file, or at runtime by providing information at db2haicu prompts. See: "DB2 High Availability Instance Configuration Utility (db2haicu)" on page 78
- The DB2 cluster manager API defines a set of functions that enable the database manager to communicate configuration changes to the cluster manager. See: "DB2 cluster manager API" on page 113

High availability through log shipping

Log shipping is the process of copying whole log files to a standby machine either from an archive device, or through a user exit program running against the primary database.

The standby database is continuously rolling forward through the log files produced by the production machine. When the production machine fails, a failover occurs and the following takes place:

- The remaining logs are transferred over to the standby machine.
- The standby database rolls forward to the end of the logs and stops.
- The clients reconnect to the standby database and resume operations.

The standby machine has its own resources (for example, disks), but must have the same physical and logical definitions as the production database. When using this approach the primary database is restored to the standby machine, by using restore utility or the split mirror function.

To ensure that you are able to recover your database in a disaster recovery situation consider the following:

- The archive location should be geographically separate from the primary site.
- Remotely mirror the log at the standby database site
- Use a synchronous mirror for no loss support. You can do this using modern disk subsystems such as ESS and EMC, or another remote mirroring technology. NVRAM cache (both local and remote) is also recommended to minimize the performance impact of a disaster recovery situation.

Note:

1. When the standby database processes a log record indicating that an index rebuild took place on the primary database, the indexes on the standby server

are not automatically rebuilt. The index will be rebuilt on the standby server either at the first connection to the database, or at the first attempt to access the index after the standby server is taken out of rollforward pending state. It is recommended that the standby server be resynchronized with the primary server if any indexes on the primary server are rebuilt. You can enable indexes to be rebuilt during rollforward operations if you set the LOGINDEXBUILD database configuration parameter.

2. If the load utility is run on the primary database with the COPY YES option specified, the standby database must have access to the copy image.
3. If the load utility is run on the primary database with the COPY NO option specified, the standby database should be resynchronized, otherwise the table space will be placed in restore pending state.
4. There are two ways to initialize a standby machine:
   a. By restoring to it from a backup image.
   b. By creating a split mirror of the production system and issuing the db2inidb command with the STANDBY option.

   Only after the standby machine has been initialized can you issue the ROLLFORWARD command on the standby system.
5. Operations that are not logged will not be replayed on the standby database. As a result, it is recommended that you re-sync the standby database after such operations. You can do this through online split mirror and suspended I/O support.

Log mirroring

IBM Data Server supports log mirroring at the database level. Mirroring log files helps protect a database from: accidental deletion of an active log; and data corruption caused by hardware failure.

If you are concerned that your active logs might be damaged (as a result of a disk crash), you should consider using the MIRRORLOGPATH configuration parameter to specify a secondary path for the database to manage copies of the active log, mirroring the volumes on which the logs are stored.

The MIRRORLOGPATH configuration parameter allows the database to write an identical second copy of log files to a different path. It is recommended that you place the secondary log path on a physically separate disk (preferably one that is also on a different disk controller). That way, the disk controller cannot be a single point of failure.

When MIRRORLOGPATH is first enabled, it will not actually be used until the next database startup. This is similar to the NEWLOGPATH configuration parameter.

If there is an error writing to either the active log path or the mirror log path, the database will mark the failing path as “bad”, write a message to the administration notification log, and write subsequent log records to the remaining “good” log path only. DB2 will not attempt to use the “bad” path again until the current log file is either full or truncated. When DB2 needs to open the next log file, it will verify that this path is valid, and if so, will begin to use it. If not, DB2 will not attempt to use the path again until the next log file is accessed for the first time. There is no attempt to synchronize the log paths, but DB2 keeps information about access errors that occur, so that the correct paths are used when log files are archived. If a failure occurs while writing to the remaining “good” path, the database shuts down.

High availability through suspended I/O and online split mirror support

IBM Data Server suspended I/O support enables you to split mirrored copies of your primary database without taking the database offline. You can use this to very quickly create a standby database to take over if the primary database fails.

Disk mirroring is the process of writing data to two separate hard disks at the same time. One copy of the data is called a mirror of the other. Splitting a mirror is the process of separating the two copies.

You can use disk mirroring to maintain a secondary copy of your primary database. You can use IBM Data Server suspended I/O functionality to split the primary and secondary mirrored copies of the database without taking the database offline. Once the primary and secondary databases copies are split, the secondary database can take over operations if the primary database fails.

If you would rather not back up a large database using the IBM Data Server backup utility, you can make copies from a mirrored image by using suspended I/O and the split mirror function. This approach also:
- Eliminates backup operation overhead from the production machine
- Represents a fast way to clone systems
- Represents a fast implementation of idle standby failover. There is no initial restore operation, and if a rollforward operation proves to be too slow, or encounters errors, reinitialization is very fast.

The db2inidb command initializes the split mirror so that it can be used:
- As a clone database
- As a standby database
- As a backup image

This command can only be issued against a split mirror, and it must be run before the split mirror can be used.

In a partitioned database environment, you do not have to suspend I/O writes on all database partitions simultaneously. You can suspend a subset of one or more database partitions to create split mirrors for performing offline backups. If the catalog partition is included in the subset, it must be the last database partition to be suspended.

In a partitioned database environment, the db2inidb command must be run on every database partition before the split image from any of the database partitions can be used. The tool can be run on all database partitions simultaneously using the db2_all command. If; however, you are using the RELOCATE USING option, you cannot use the db2_all command to run db2inidb on all of the database partitions simultaneously. A separate configuration file must be supplied for each database partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being changed, every database partition will be affected and the db2relocatedb command must be run with a separate configuration file on each database partition. If containers belonging to a single database partition are being moved, the db2relocatedb command only needs to be run once on that database partition.

Note: Ensure that the split mirror contains all containers and directories which comprise the database, including the volume directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.

Chapter 4. Configuring for high availability

To configure your DB2 database solution for high availability, you must: schedule database maintenance activities; configure the primary and standby database servers to know about each other and their respective roles in the event of a failure; and configure any cluster managing software to transfer workload from a failed cluster node.

Before configuring your database solution:
- Assemble and install the underlying hardware and software components that make up the solution. These underlying components might include: power supply; network connectivity; network cards; disks or other storage devices; operating systems; and cluster managing software.
- Test these underlying components without any database workload to make sure they are functioning properly before attempting to use them in database load balancing, failover, or recovery operations.

Redundancy is an important part of a high availability solution. However, if you do not schedule maintenance wisely, if you run out of storage space for needed recovery logs, or if your cluster managing software is not configured correctly, your solution might not be available when your users need to do crucial work with the database.

Configuring for high availability includes:
- Configure client reroute

  "Automatic client reroute description and setup"
- Configure fault monitor

  "DB2 fault monitor registry file" on page 28
- Configure DB2 High Availability Disaster Recovery

  "Initializing high availability disaster recovery (HADR)" on page 31
- Schedule maintenance activities

  "Scheduling maintenance for high availability" on page 53
- Configure logging

  "Configuring database logging options" on page 56
- Configure cluster managing software

  "Configuring a Clustered environment for high availability" on page 68

Automatic client reroute description and setup

The main goal of the automatic client reroute feature is to enable an IBM Data Server Client application to recover from a loss of communications so that the application can continue its work with minimal interruption. As the name suggests, rerouting is central to the support of continuous operations. But rerouting is only possible when there is an alternate location that is identified to the client connection.

The automatic client reroute feature could be used within the following configurable environments if the server is DB2 on Linux, UNIX, or Windows:
1. Enterprise Server Edition (ESE) with the database partitioning feature (DPF)

2. WebSphere Replication Server
3. High availability cluster multiprocessor (HACMP)
4. High availability disaster recovery (HADR).

   Automatic client reroute works in conjunction with HADR to allow a client application to continue its work with minimal interruption after a failover of the database being accessed.

The automatic client reroute feature could also be used in the following configuration if the database server is on System i® or System z®:

1. IBM Data Server Client connects to a z/OS or i5/OS® system through a DB2 Connect server which has an alternate server. The automatic client reroute is used between the IBM Data Server Client and two DB2 Connect servers.
2. DB2 Connect personal or server products accessing a DB2 for z/OS Sysplex data sharing environment. Automatic client reroute is used between DB2 Connect and the z/OS Sysplex system. The automatic client reroute feature supports seamless failover between a DB2 Connect-licensed client and the Sysplex. For more information about seamless failover, see the topic about automatic client reroute (client-side) in the DB2 Information Center.

In the case of the DB2 Connect server and its alternate, because there is no requirement for the synchronization of local databases, you only need to ensure that both the original and alternate DB2 Connect servers have the target host or System i database catalogued in such a way that it is accessible using an identical database alias.

In order for the DB2 database system to have the ability to recover from a loss of communications, an alternative server location must be specified before the loss of communication occurs. The UPDATE ALTERNATE SERVER FOR DATABASE command is used to define the alternate server location on a particular database.

After you have specified the alternate server location on a particular database at the server instance, the alternate server location information is returned to the IBM Data Server Client as part of the connection process. In the case of using automatic client reroute between DB2 Connect personal or server products and a host or System i database server, the remote server must provide one or more alternate addresses for itself. In the case of DB2 for z/OS, multiple addresses are known if the database is a Sysplex data sharing environment. Therefore an alternate server does not need to be cataloged on DB2 Connect. If communication between the client and the server is lost for any reason, the IBM Data Server Client will attempt to reestablish the connection by using the alternate server information. The IBM Data Server Client will attempt to reconnect with a database server which could be the original server, and alternate server listed in the database directory file at the server, or an alternate server that is in the server list returned by the z/OS Sysplex system. The timing of these attempts to reestablish a connection varies from very rapid attempts initially to a gradual lengthening of the intervals between the attempts.

Once a connection is successful, the SQLCODE -30108 is returned to indicate that a database connection has been reestablished following the communication failure. The hostname or IP address and service name or port number are returned. The IBM Data Server Client only returns the error for the original communications failure to the application if the reestablishment of the client communications is not possible to either the original or alternative server.

The following considerations involving alternate server connectivity in a DB2 Connect server environment should also be noted:

- When using a DB2 Connect server for providing access to a host or System i database on behalf of both remote and local clients, confusion can arise regarding alternate server connectivity information in a system database directory entry. To minimize this confusion, consider cataloging two entries in the system database directory to represent the same host or System i database. Catalog one entry for remote clients and catalog another for local clients.
- Any SYSPLEX information that is returned from a target DB2 for z/OS server is kept only in cache at the DB2 Connect server. Only one alternate server is written to disk. When multiple alternates or multiple active servers exist, the information is only maintained in memory and is lost when the process terminates.

In general, if an alternate server is specified, automatic client reroute will be enabled when a communication error (sqlcode -30081) or a sqlcode -1224 is detected. However, in a high availability disaster recovery (HADR) environment, it will also be enabled if sqlcode -1776 is returned back from the HADR standby server.

Configuring automatic client reroute retry behavior using registry variables

When a primary database server fails, DB2 automatic client reroute transfers client application requests to a secondary database server. DB2 client reroute then repeatedly tries to reconnect to the primary database. You can configure the maximum number of reconnection attempts DB2 client reroute makes, and the sleep time between consecutive reconnection attempts.

By default, the automatic client reroute feature retries the connection to a database repeatedly for up to 10 minutes. It is, however, possible to configure the exact retry behavior using one or both of the following two registry variables:

- DB2_MAX_CLIENT_CONNRETRIES: The maximum number of connection retries attempted by automatic client reroute.
- DB2_CONNRETRIES_INTERVAL: The sleep time between consecutive connection retries, in number of seconds.

If DB2_MAX_CLIENT_CONNRETRIES is set, but DB2_CONNRETRIES_INTERVAL is not, DB2_CONNRETRIES_INTERVAL defaults to 30.

If DB2_MAX_CLIENT_CONNRETRIES is not set, but DB2_CONNRETRIES_INTERVAL is set, DB2_MAX_CLIENT_CONNRETRIES defaults to 10.

If neither DB2_MAX_CLIENT_CONNRETRIES nor DB2_CONNRETRIES_INTERVAL is set, the automatic client reroute feature reverts to its default behavior described previously.

Note:

- Users of Type 4 connectivity with the DB2 Universal JDBC Driver should use the following two datasource properties to configure automatic client rerouting:
  - maxRetriesForClientReroute: Use this property to limit the number of retries if the primary connection to the server fails. This property is only used if the retryIntervalClientReroute property is also set.

  - retryIntervalForClientReroute: Use this property to specify the amount of time (in seconds) to sleep before retrying again. This property is only used if the maxRetriesForClientReroute property is also set.
- The settings DB2_MAX_CLIENT_CONNRETRIES and DB2_CONNRETRIES_INTERVAL have no effect when automatic client reroute is enabled for a client accessing a DB2 for z/OS Sysplex (when enableAcr is True in the db2dsdriver configuration file). For more information, see the topic about automatic client reroute (client-side) in the DB2 Information Center.

Using client connection timeout with automatic client reroute

In a standard DB2 database solution implementation, a client application attempts to connect to a database server until successful, or until a configured connection timeout length of time has passed. If the connection times out, a communications error is returned to the application. If you are using DB2 automatic client reroute, DB2 will redirect that client application connection will to an alternate database server instead of generating a communications error.

For CLI/ODBC, OLE DB, and ADO.NET applications, you can set a connection timeout value to specify the number of seconds that the client application waits for a reply when trying to establish a connection to a server before terminating the connection attempt and generating a communication timeout.

If client reroute is enabled, you need to set the connection timeout value to a value that is equal to or greater than the maximum time it takes to connect to the server. Otherwise, the connection might timeout and be rerouted to the alternate server by client reroute. For example, if on a normal day it takes about 10 seconds to connect to the server, and on a busy day it takes about 20 seconds, the connection timeout value should be set to at least 20 seconds.

Configuring automatic client reroute for client connection distributor technology

Distributor or dispatcher technologies such as WebSphere EdgeServer distribute client application reconnection requests to a defined set of systems if a primary database server fails. If you are using distributor technology with DB2 automatic client reroute, you must identify the distributor itself as the alternate server to DB2 automatic client reroute.

You might be using distributor technology in an environment similar to the following:

Client —> distributor technology —> (DB2 Connect Server 1 or DB2 Connect Server 2) —> DB2 z/OS

where:
- The distributor technology component has a TCP/IP host name of DThostname
- The DB2 Connect Server 1 has a TCP/IP host name of GWYhostname1
- The DB2 Connect Server 2 has a TCP/IP host name of GWYhostname2
- The DB2 z/OS server has a TCP/IP host name of zOShostname

The client is catalogued using **DThostname** in order to utilize the distributor technology to access either of the DB2 Connect Servers. The intervening distributor technology makes the decision to use **GWYhostname1** or **GWYhostname2**. Once the decision is made, the client has a direct socket connection to one of these two

DB2 Connect gateways. Once the socket connectivity is established to the chosen DB2 Connect server, you have a typical client to DB2 Connect server to DB2 z/OS connectivity.

For example, assume the distributor chooses **GWYhostname2**. This produces the following environment:

Client —> DB2 Connect Server 2 —> DB2 z/OS

The distributor does not retry any of the connections if there is any communication failure. If you want to enable the automatic client reroute feature for a database in such an environment, the alternative server for the associated database or databases in the DB2 Connect server (DB2 Connect Server 1 or DB2 Connect Server 2) should be set up to be the distributor (DThostname). Then, if DB2 Connect Server 1 locks up for any reason, automatic client rerouting is triggered and a client connection is retried with the distributor as both the primary and the alternate server. This option allows you to combine and maintain the distributor capabilities with the DB2 automatic client reroute feature. Setting the alternate server to a host other than the distributor host name still provides the clients with the automatic client reroute feature. However, the clients will establish direct connections to the defined alternate server and bypass the distributor technology, which eliminates the distributor and the value that it brings.

The automatic client reroute feature intercepts the following SQL codes:
- sqlcode -20157
- sqlcode -1768 (reason code = 7)

Note: Client reroute might not be informed of socket failures in a timely fashion if the setting of the ″TCP Keepalive″ operating system configurations parameter is too high. (Note that the name of this configuration parameter varies by platform.)

Identifying an alternate server for automatic client reroute

Whenever a DB2 server or DB2 Connect server crashes, each client that is connected to that server receives a communications error which terminates the connection resulting in an application error. In cases where availability is important, you should have implemented either a redundant set up or the ability to fail the server over to a standby node. In either case, the DB2 client code attempts to re-establish the connection to the original server which might be running on a failover node (the IP address fails over as well), or to a new server.

To define a new or alternate server:

Use the **UPDATE ALTERNATE SERVER FOR DATABASE** or **UPDATE ALTERNATE SERVER FOR LDAP DATABASE** command.
These commands update the alternate server information for a database alias in the system database directory.

Client reroute setup when using IBM Data Server Driver for JDBC and SQLJ

Whenever a server crashes, each client that is connected to that server receives a communication error, which terminates the connection and results in an application error. When availability is important, you should have a redundant setup or failover support. Failover is the ability of a server to take over operations when another server fails. When the connection is terminated, the IBM Data Server

Driver for JDBC and SQLJ client attempts to re-establish the connection to the original server (which might be running on a failover node) or to a new server. When the connection is re-established, the application receives an `SQLException` that informs it of the transaction failure, but the application can continue with the next transaction.

For detailed information, refer to the topic about client reroute support for the IBM Data Server Driver for JDBC and SQLJ in the Related Concepts section, below.

Automatic client reroute limitations

Consider DB2 database client reroute restrictions when designing your high availability DB2 database solution.

Here is a list of limitations of the DB2 database automatic client reroute feature:

- Automatic client reroute is only supported when the communications protocol used for connecting to the DB2 database server, or to the DB2 Connect server, is TCP/IP. This means that if the connection is using a different protocol other than TCP/IP, the automatic client reroute feature will not be enabled. Even if DB2 database is set up for a loopback, TCP/IP communications protocol must be used in order accommodate the automatic client reroute feature.
- When using automatic reroute between the DB2 Connect personal or server products and a host or System i database server, if you are in the following situations you will have the associated implications:
  - When using a DB2 Connect server for providing access to a host or System i database on behalf of both remote and local clients, confusion can arise regarding alternate server connectivity information in a system database directory entry. To minimize this confusion, consider cataloging two entries in the system database directory to represent the same host or System i database. Catalog one entry for remote clients and catalog another for local clients.
  - Any SYSPLEX information that is returned from a target DB2 for z/OS server is kept only in cache at the DB2 Connect server. Only one alternate server is written to disk. When multiple alternates or multiple active servers exist, the information is only maintained in memory and is lost when the process terminates.
- If the connection is reestablished to the alternate server location, any new connection to the same database alias will be connected to the alternate server location. If you want any new connection to be established, to the original location in case the problem on the original location is fixed, there are a couple of options from which to choose:
  - You need to take the alternate server offline and allow the connections to fail back over to the original server. (This assumes that the original server has been cataloged using the UPDATE ALTERNATE SERVER command such that it is set to be the alternate location for the alternate server.)
  - You could catalog a new database alias to be used by the new connections.
  - You could uncatalog the database entry and re-catalog it again.
- DB2 Database for Linux, UNIX, and Windows supports the automatic client reroute feature for both the client and the server if both the client and server support this feature. Other DB2 database product families do not currently support this feature.
- The behavior of the automatic client reroute feature and the behavior of the automatic client rerouting in a DB2 for z/OS sysplex environment are somewhat different. Specifically:

  - The automatic client reroute feature requires the primary server to designate a single alternative server. This is done using the UPDATE ALTERNATE SERVER FOR DATABASE or UPDATE ALTERNATE SERVER FOR LDAP DATABASE command issued at the primary server. This command updates the local database directory with the alternate server information so that other applications at the same client have access this information. By contrast, a data-sharing sysplex used for DB2 for z/OS maintains, in memory, a list of one or more servers to which the client can connect. If a communication failure happens, the client uses that list of servers to determine the location of the appropriate alternative server.
  - In the case of the automatic client reroute feature, the server informs the client of the most current special register settings whenever a special register setting is changed. This allows the client, to the best of its ability, to reestablish the runtime environment after a reroute has occurred. By contrast, a Sysplex used for DB2 for z/OS returns the special register settings to the client on commit boundaries therefore any special registers changed within the unit of work that has been rerouted need to be replayed. All others will be replayed automatically.

  As of DB2® Universal Database™ Version 8 FixPak 7, full automatic client reroute support is available only between a Linux, UNIX, or Windows client and a Linux, UNIX, or Windows server. It is not available between a Linux, UNIX, or Windows client and a DB2 for z/OS Sysplex server (any supported version); only the reroute capability is supported.
- The DB2 database server installed in the alternate host server must be the same version (but could have a higher FixPak) when compared to the DB2 database instance installed on the original host server.
- Regardless of whether you have authority to update the database directory at the client machine, the alternate server information is always kept in memory. In other words, if you did not have authority to update the database directory (or because it is a read-only database directory), other applications will not be able to determine and use the alternate server, because the memory is not shared among applications.
- The same authentication is applied to all alternate locations. This means that the client will be unable to reestablish the database connection if the alternate location has a different authentication type than the original location.
- When there is a communication failure, all session resources such as global temporary tables, identity, sequences, cursors, server options (SET SERVER OPTION) for federated processing and special registers are all lost. The application is responsible to reestablish the session resources in order to continue processing the work. You do not have to run any of the special register statements after the connection is reestablished, because the DB2 database will replay the special register statements that were issued before the communication error. However, some of the special registers will not be replayed. They are:
  - SET ENCRYPTPW
  - SET EVENT MONITOR STATE
  - SET SESSION AUTHORIZATION
  - SET TRANSFORM GROUP

  When you have a problem with DB2 Connect, you should refer to the list of restricted special registers specific to the DB2 Connect product on a data server.
- If, after the connection is reestablished following a communication failure and the client is using CLI, JCC Type 2 or Type 4 drivers, then those SQL and XQuery statements that have been prepared against the original server are

implicitly re-prepared with the new server. However, embedded SQL routines (for example, SQC or SQX applications), are not re-prepared with the new server.

- Do not run high availability disaster recovery (HADR) commands on client reroute-enabled database aliases. HADR commands are implemented to identify the target database using database aliases. Consequently, if the target database has an alternative database defined, it is difficult for HADR commands to determine the database on which the command is actually operating. While a client might need to connect using a client reroute-enabled alias, HADR commands must be applied on a specific database. To accommodate this, you can define aliases specific to the primary and standby databases and only run HADR commands on those aliases.

An alternate way to implement automatic client rerouting is to use the DNS entry to specify an alternate IP address for a DNS entry. The idea is to specify a second IP address (an alternate server location) in the DNS entry; the client would not know about an alternate server, but at connect time DB2 database system would alternate between the IP addresses for the DNS entry.

DB2 fault monitor registry file

A fault monitor registry file is created for every DB2 database manager instance on each physical machine when the fault monitor daemon is started. The keywords and values in this file specify the behavior of the fault monitors.

The fault monitor registry file can be found in the `/sqllib/` directory and is called `fm.<machine_name>.reg`. This file can be altered using the db2fm command.

If the fault monitor registry file does not exist, the default values will be used.

Here is an example of the contents of the fault monitor registry file:

```
FM_ON = no
FM_ACTIVE = yes
START_TIMEOUT = 600
STOP_TIMEOUT = 600
STATUS_TIMEOUT = 20
STATUS_INTERVAL = 20
RESTART_RETRIES = 3
ACTION_RETRIES = 3
NOTIFY_ADDRESS = <instance_name>@<machine_name>
```

Fault monitor registry file keywords

FM_ON

Specifies whether or not the fault monitor should be started. If the value is set to `NO`, the fault monitor daemon will not be started, or will be turned off if it had already been started. The default value is `NO`.

FM_ACTIVE

Specifies whether or not the fault monitor is active. The fault monitor will only take action if both FM_ON and FM_ACTIVE are set to `YES`. If FM_ON is set to `YES` and FM_ACTIVE is set to `NO`, the fault monitor daemon will be started, but it will not be active. That means that is will not try to bring DB2 back online if it shuts down. The default value is `YES`.

START_TIMEOUT

Specifies the amount of time within which the fault monitor must start the service it is monitoring. The default value is 600 seconds.

STOP_TIMEOUT

Specifies the amount of time within which the fault monitor must bring down the service it is monitoring. The default value is 600 seconds.

STATUS_TIMEOUT

Specifies the amount of time within which the fault monitor must get the status of the service it is monitoring. The default value is 20 seconds.

STATUS_INTERVAL

Specifies the minimum time between two consecutive calls to obtain the status of the service that is being monitored. The default value is 20 seconds.

RESTART_RETRIES

Specifies the number of times the fault monitor will try to obtain the status of the service being monitored after a failed attempt. Once this number is reached the fault monitor will take action to bring the service back online. The default value is 3.

ACTION_RETRIES

Specifies the number of times the fault monitor will attempt to bring the service back online. The default value is 3.

NOTIFY_ADDRESS

Specifies the e-mail address to which the fault monitor will send notification messages. The default is <instance_name>@<machine_name>)

Configuring DB2 fault monitor using the db2fm command

You can alter the DB2 fault monitor registry file using the db2fm command.

Here are some examples of using the db2fm command to update the fault monitor registry file:

Example 1: Update START_TIMEOUT

To update the START_TIMEOUT value to 100 seconds for instance DB2INST1, type the following command from a DB2 database command window:

```
db2fm -i db2inst1 -T 100
```

Example 2: Update STOP_TIMEOUT

To update the STOP_TIMEOUT value to 200 seconds for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -T /200
```

Example 3: Update START_TIMEOUT and STOP_TIMEOUT

To update the START_TIMEOUT value to 100 seconds and the STOP_TIMEOUT value to 200 seconds for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -T 100/200
```

Example 4: Turn on fault monitoring

To turn on fault monitoring for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -f yes
```

Example 5: Turn off fault monitoring

To turn off fault monitoring for instance DB2INST1, type the following command:

```
db2fm -i db2inst1 -f no
```

To confirm that fault monitor is no longer running for DB2INST1, type the following command on UNIX systems:

```
ps -ef|grep -i fm
```

On Linux, type the following command:

```
ps auxw|grep -i fm
```

An entry that shows db2fmd and DB2INST1 indicates that the fault monitor is still running on that instance. To turn off the fault monitor, type the following command as the instance owner:

```
db2fm -i db2inst1 -D
```

Configuring DB2 fault monitor using db2fmc and system commands

You can configure the DB2 fault monitor using the DB2 Fault Monitor Controller Utility (FMCU) command db2fmcu or system commands.

Here are some examples of using db2fmcu and system commands to configure the fault monitor:

Example 1: Prevent FMC from being launched

You can prevent the FMC from being launched by using the DB2 Fault Monitor Controller Utility (FMCU). The FMCU must be run as root because it accesses the system's inittab file. To block the FMC from being run, type the following command as root:

```
db2fmcu -d
```

Note: If you apply a DB2 Data Server fix pack this will be reset so that the inittab will again be configured to include the FMC. To prevent the FMC from being launched after you have applied a fix pack, you must reissue the above command.

Example 2: Include FMC to be launched

To reverse the `db2fmcu -d` command and reconfigure the inittab to include the FMC, type the following command:

```
db2fmcu -u -p  <fullpath>
```

where <fullpath> is the complete path to the db2fmcd object, for example /opt/IBM/db2/bin/db2fmcd.

Example 3: automatically start the DB2 database manager instance

You can also enable FMC to automatically start the instance when the system is first booted. To enable this feature for instance DB2INST1, type the following command:

```
db2iauto -on db2inst1
```

Example 4: disable automatically starting the instance

To turn off the autostart behaviour, type the following command:

```
db2iauto -off db2inst1
```

Example 5: prevent fault monitor processes from being launched

You can also prevent fault monitor processes from being launched for a specific instances on the system by changing a field in the global registry record for the instance. To change the global registry field to disable fault monitors for instance DB2INST1, type the following command as root:

```
db2greg -updinstrec instancename=db2inst1!startatboot=0
```

To reverse this command and re-enable fault monitors for instance DB2INST1, type the following command as root:

```
db2greg -updinstrec instancename=db2inst1!startatboot=1
```

Initializing high availability disaster recovery (HADR)

Use the following procedure to set up and initialize the primary and standby databases for DB2 High Availability Disaster Recovery (HADR).

HADR can be initialized through the command line processor (CLP), the Set Up High Availability Disaster Recovery (HADR) wizard in the Control Center, or by calling the db2HADRStart API.

To use the CLP to initialize HADR on your system for the first time:

1. Determine the host name, host IP address, and the service name or port number for each of the HADR databases.

   If a host has multiple network interfaces, ensure that the HADR host name or IP address maps to the intended one. You need to allocate separate HADR ports in /etc/services for each protected database. These cannot be the same as the ports allocated to the instance. The host name can only map to one IP address.

   Note: The instance names for the primary and standby databases do not have to be the same.
2. Create the standby database by restoring a backup image or by initializing a split mirror, based on the existing database that is to be the primary.

   In the following example, the BACKUP DATABASE and RESTORE DATABASE commands are used to initialize database SOCKS as a standby database. In this case, an NFS mounted file system is accessible at both sites.

   Issue the following command at the primary database:

   ```
   backup db socks to /nfs1/backups/db2/socks
   ```

   Issue the following command at the standby database:

   ```
   restore db socks from /nfs1/backups/db2/socks replace history file
   ```

   The following example illustrates how to use the db2inidb utility to initialize the standby database using a split mirror of the primary database. This procedure is an alternative to the backup and restore procedure illustrated above.

   Issue the following command at the standby database:

   ```
   db2inidb socks as standby
   ```

   Note:

a. The database names for the primary and standby databases must be the same.
b. It is recommended that you do not issue the ROLLFORWARD DATABASE command on the standby database after the restore operation or split mirror initialization. The results of using a rollforward operation might differ slightly from replaying the logs using HADR on the standby database. If the databases are not identical, issuing the START HADR command with the AS STANDBY option will fail.
c. When using the RESTORE DATABASE command, it is recommended that the REPLACE HISTORY FILE option is used.
d. When creating the standby database using the RESTORE DATABASE command, you must ensure that the standby remains in rollforward mode. This means that you cannot issue the ROLLFORWARD DATABASE command with either the COMPLETE option or the STOP option. An error will be returned if the START HADR command with the AS STANDBY option is attempted on the database after rollforward is stopped.
e. The following RESTORE DATABASE command options should be avoided when setting up the standby database: TABLESPACE, INTO, REDIRECT, and WITHOUT ROLLING FORWARD.
f. When setting up the standby database using the db2inidb utility, do not use the SNAPSHOT or MIRROR options. You can specify the RELOCATE USING option to change one or more of the following configuration attributes: instance name, log path, and database path. However, you must not change the database name or the table space container paths.

3. Set the HADR configuration parameters on the primary and standby databases.

   Note: It is very important that you set the following configuration parameters after the standby databases has been created:
   - HADR_LOCAL_HOST
   - HADR_LOCAL_SVC
   - HADR_REMOTE_HOST
   - HADR_REMOTE_SVC
   - HADR_REMOTE_INST

   If they are set prior to creating the standby database, the settings on the standby database will reflect what is set on the primary database.
4. Connect to the standby instance and start HADR on the standby database, as in the following example:
   ```
   START HADR ON DB SOCKS AS STANDBY
   ```

   Note: Usually, the standby database is started first. If you start the primary database first, this startup procedure will fail if the standby database is not started within the time period specified by the HADR_TIMEOUT database configuration parameter.
5. Connect to the primary instance and start HADR on the primary database, as in the following example:
   ```
   START HADR ON DB SOCKS AS PRIMARY
   ```
6. HADR is now started on the primary and standby databases.

   To open the Set Up High Availability Disaster Recovery (HADR) Databases wizard:
   a. From the Control Center expand the object tree until you find the database for which you want to configure HADR.

b. Right-click the database and click High Availability Disaster Recovery → Set Up in the pop-up menu. The Set Up High Availability Disaster Recovery Databases wizard opens.

Additional information is provided through the contextual help facility within the Control Center.

Note: You can start HADR within the Set Up High Availability Disaster Recovery Databases wizard, or you can just use the wizard to initialize HADR, then start it at another time. To open the Start HADR window:

a. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.

b. Click Start HADR. The Start HADR window opens.

Configuring automatic client reroute and High Availability Disaster Recovery (HADR)

You can use the automatic client reroute feature with the High Availability Disaster Recovery (HADR) feature to transfer client application requests from a failed database server to a standby database server.

Restrictions

- Rerouting is only possible when an alternate database location has been specified at the server.
- Automatic client reroute is only supported with TCP/IP protocol.

Configuration details

- Use the UPDATE ALTERNATE SERVER FOR DATABASE command to enable automatic client reroute.
- Client reroute is enabled by default if you set up HADR using the Set Up High Availability Disaster Recovery (HADR) Databases wizard in the Control Center.
- Automatic client reroute does not use the HADR_REMOTE_HOST and HADR_REMOTE_SVC database configuration parameters.
- The alternate host location is stored in the system database directory file at the server.
- If automatic client reroute is not enabled, client applications will receive error message SQL30081, and no further attempts will be made to establish a connection with the server.

Using the UPDATE ALTERNATE SERVER FOR DATABASE command to set up automatic client reroute with HADR

Your system is set up as follows:

- You have a client where database MUSIC is catalogued as being located at host HORNET.
- Database MUSIC is the primary database and its corresponding standby database, also MUSIC, resides on host MONTERO with port number 456, which is assigned by the SVCENAME configuration parameter.

To enable automatic client reroute, update the alternate server for database MUSIC on host HORNET:

```
db2 update alternate server for database music using hostname montero port 456
```

After this command is issued, the client must successfully connect to host HORNET to obtain the alternate server information. Then, if a communication error occurs between the client and database MUSIC at host HORNET, the client will first attempt to reconnect to database MUSIC at host HORNET. If this fails, the client will then attempt to establish a connection with the standby database MUSIC on host MONTERO.

Index logging and high availability disaster recovery (HADR)

You should consider setting the database configuration parameters LOGINDEXBUILD and INDEXREC for DB2 High Availability Disaster Recovery (HADR) databases.

Using the LOGINDEXBUILD database configuration parameter

Recommendation: For HADR databases, set the LOGINDEXBUILD database configuration parameter to ON to ensure that complete information is logged for index creation, recreation, and reorganization. Although this means that index builds might take longer on the primary system and that more log space is required, the indexes will be rebuilt on the standby system during HADR log replay and will be available when a failover takes place. If index builds on the primary system are not logged and a failover occurs, any invalid indexes that remain after the failover is complete will have to be rebuilt before they can be accessed. While the indexes are being recreated, they cannot be accessed by any applications.

Note: If the LOG INDEX BUILD table attribute is set to its default value of NULL, DB2 will use the value specified for the LOGINDEXBUILD database configuration parameter. If the LOG INDEX BUILD table attribute is set to ON or OFF, the value specified for the LOGINDEXBUILD database configuration parameter will be ignored.

You might choose to set the LOG INDEX BUILD table attribute to OFF on one or more tables for either of the following reasons:
- You do not have enough active log space to support logging of the index builds.
- The index data is very large and the table is not accessed often; therefore, it is acceptable for the indexes to be recreated at the end of the takeover operation. In this case, set the INDEXREC configuration parameter to RESTART. Because the table is not frequently accessed, this setting will cause the system to recreate the indexes at the end of the takeover operation instead of waiting for the first time the table is accessed after the takeover operation.

If the LOG INDEX BUILD table attribute is set to OFF on one or more tables, any index build operation on those tables might cause the indexes to be recreated any time a takeover operation occurs. Similarly, if the LOG INDEX BUILD table attribute is set to its default value of NULL, and the LOGINDEXBUILD database configuration parameter is set to OFF, any index build operation on a table might cause the indexes on that table to be recreated any time a takeover operation occurs. You can prevent the indexes from being recreated by taking one of the following actions:

- After all invalid indexes are recreated on the new primary database, take a backup of the database and apply it to the standby database. As a result of doing this, the standby database does not have to apply the logs used for recreating invalid indexes on the primary database, which would mark those indexes as rebuild required on the standby database.
- Set the LOG INDEX BUILD table attribute to ON, or set the LOG INDEX BUILD table attribute to NULL and the LOGINDEXBUILD configuration parameter to ON on the standby database to ensure that the index recreation will be logged.

Using the INDEXREC database configuration parameter

Recommendation: Set the INDEXREC database configuration parameter to RESTART (the default) on both the primary and standby databases. This will cause invalid indexes to be rebuilt after a takeover operation is complete. If any index builds have not been logged, this setting allows DB2 to check for invalid indexes and to rebuild them. This process takes place in the background, and the database will be accessible after the takeover operation has completed successfully.

If a transaction accesses a table that has invalid indexes before the indexes have been rebuilt by the background recreate index process, the invalid indexes will be rebuilt by the first transaction that accesses it.

Database configuration for high availability disaster recovery (HADR)

You can use database configuration parameters to achieve optimal performance with DB2 High Availability Disaster Recovery (HADR).

To achieve optimal performance with DB2 High Availability Disaster Recovery (HADR), ensure that your database configuration meets the following requirements.

Recommendation: To the extent possible, the database configuration parameters and database manager configuration parameters should be identical on the systems where the primary and standby databases reside. If the configuration parameters are not properly set on the standby database the following problems might occur:

- Error messages might be returned on the standby database while replaying the log files that were shipped from the primary database.
- After a takeover operation, the new primary database will not be able to handle the workload, resulting in performance problems or in applications receiving error messages they were not receiving when they were connected to the original primary database.

Changes to the configuration parameters on the primary database are not automatically propagated to the standby database and must be made manually on the standby database. For dynamic configuration parameters, changes will take effect without shutting down and restarting the database management system (DBMS) or the database. For non-dynamic configuration parameters, changes will take effect after the standby database is restarted.

Size of log files configuration parameter on the standby database

One exception to the configuration parameter behavior described above is the LOGFILSIZ database configuration parameter. Although this parameter is not replicated to the standby database, to guarantee identical log files on both databases, the standby database ignores the local LOGFILSIZ configuration and creates local log files that match the size of the log files on the primary database.

After a takeover, the original standby (new primary) will keep using the value that was set on the original primary until the database is restarted. At that point, the new primary will revert to the value configured locally. In addition, the new primary also truncates the current log file and resizes any pre-created log files.

If the databases keep switching roles via non-forced takeover and neither database is deactivated, then the log file size used will always be the one established by the very first primary. However, if there is a deactivate and then a restart on the original standby (new primary) then it would use the log file size configured locally. This log file size would continue to be used if the original primary takes over again. Only after a deactivate and restart on the original primary would the log file size revert back to the settings on the original primary.

Log receive buffer size on the standby database

By default, the log receive buffer size on the standby database will be two times the value specified for the LOGBUFSZ configuration parameter on the primary database. There might be times when this size is not sufficient. For example, when the HADR synchronization mode is asynchronous and the primary and standby databases are in peer state, if the primary database is experiencing a high transaction load, the log receive buffer on the standby database might fill to capacity and the log shipping operation from the primary database might stall. To manage these temporary peaks, you can increase the size of the log receive buffer on the standby database by modifying the DB2_HADR_BUF_SIZE registry variable.

Load operations and HADR

If a load operation is executed on the primary database with the COPY YES option, the command will execute on the primary database and the data will be replicated to the standby database as long as the copy can be accessed through the path or device specified by the LOAD command. If the standby database cannot access the data, the table space in which the table is stored is marked invalid on the standby database. The standby database will skip future log records that pertain to this table space. To ensure that the load operation can access the copy on the standby database, it is recommended that you use a shared location for the output file from the COPY YES option. Alternatively, you can deactivate the standby database while the load operation is performed, perform the load on the primary, place a copy of the output file in the standby path, and then activate the standby database.

If a load operation is executed on the primary database with the NONRECOVERABLE option, the command will execute on the primary database and the table on the standby database will be marked invalid. The standby database will skip future log records that pertain to this table. You can choose to issue the LOAD command with the COPY YES and REPLACE options specified to bring the table back, or you can drop the table to recover the space.

Since executing a load operation with the COPY NO option is not supported with HADR, the command is automatically converted to a load operation with the NONRECOVERABLE option. To enable a load operation with the COPY NO option to be converted to a load operation with the COPY YES option, set the DB2_LOAD_COPY_NO_OVERRIDE registry variable on the primary database. This registry variable is ignored by the standby database. Ensure that the device or directory specified on the primary database can be accessed by the standby database using the same path, device, or load library.

If you are using Tivoli Storage Manager (TSM) to perform a load operation with the COPY YES option, you might need to set the VENDOROPT configuration parameter on the primary and standby databases. Depending on how TSM is configured, the values on the primary and standby databases might not be the same. Also, when using TSM to perform a load operation with the COPY YES option, you must issue the db2adutl command with the GRANT option to give the standby database read access for the files that are loaded.

If table data is replicated by a load operation with the COPY YES option specified, the indexes will be replicated as follows:

- If the indexing mode is set to REBUILD and the table attribute is set to LOG INDEX BUILD, or the table attribute is set to DEFAULT and the LOGINDEXBUILD database configuration parameter is set to ON, the primary database will include the rebuilt index object in the copy file to enable the standby database to replicate the index object. If the index object on the standby database is marked invalid before the load operation, it will become usable again after the load operation as a result of the index rebuild.
- If the indexing mode is set to INCREMENTAL and the table attribute is set to LOG INDEX BUILD, or the table attribute is set to NULL and LOGINDEXBUILD database configuration parameter on the primary database is set to ON, the index object on the standby database is updated only if it is not marked invalid before the load operation. Otherwise, the index is marked invalid on the standby database.

Registry variable DB2_HADR_PEER_WAIT_LIMIT

When registry variable *DB2_HADR_PEER_WAIT_LIMIT* is set, HADR primary database will break out of peer state if logging on the primary database has been blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, primary database will break the connection to the standby database. If peer window is disabled, the primary will enter disconnected state and logging resumes. If peer window is enabled, the primary database will enter disconnected peer state, in which logging continues to be blocked. Primary leaves disconnected peer state upon re-connection or peer window expiration. Logging resumes once primary leaves disconnected peer state.

Honoring peer window transition when breaking out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies (safe takeover from standby as long as it's still in disconnected-peer state).

On the standby side, after disconnection, the database will continue replaying already received logs. Once received logs have been replayed, the standby will reconnect to the primary. Upon re-connection, normal state transition follows (first remote catchup state, then peer state)

Relationship to HADR_TIMEOUT:

Database configuration parameter **HADR_TIMEOUT** does not break the primary out of peer state if the primary keeps receiving heartbeat messages from the standby while blocked. **HADR_TIMEOUT** is a timeout for the HADR network layer. An HADR database breaks the connection to its partner database if it has not received any message from its partner for the **HADR_TIMEOUT** period. It does not control timeout for higher layer operations such as log shipping and ack. If log replay on the standby database is stuck on a large operation such as load or reorganization, the HADR component will still send heartbeat messages to the primary database on normal schedule. In such a scenario, the primary will be blocked as long as the standby replay is blocked, unless *DB2_HADR_PEER_WAIT_LIMIT* is set.

DB2_HADR_PEER_WAIT_LIMIT unblocks primary logging regardless of connection status. Note that even if *DB2_HADR_PEER_WAIT_LIMIT* is not set, primary always breaks out of peer state when a network error is detected or the connection is closed (possibly as result of **HADR_TIMEOUT**).

HADR configuration parameters

Several new database configuration parameters are available to support HADR. Setting these parameters does not change the role of a database. You must issue the START HADR or STOP HADR commands to change the role of a database.

HADR configuration parameters are not dynamic. Any changes made to an HADR configuration parameter are not effective until the database has been shut down and restarted. In a partitioned database environment, the HADR configuration parameters are visible and can be changed, but they are ignored.

The local host name of the primary database must be the same as the remote host name of the standby database, and the local host name of the standby database must be the same as the remote host name of the primary database. Use the HADR_LOCAL_HOST and HADR_REMOTE_HOST configuration parameters to set the local and remote hosts for each database. Configuration consistency for the local and remote host names is checked when a connection is established to ensure that the remote host specified is the expected database.

An HADR database can be configured to use either IPv4 or IPv6 to locate its partner database. If the host server does not support IPv6, the database will use IPv4. If the server does support IPv6, whether the database uses IPv4 or IPv6 depends upon the format of the address specified for the HADR_LOCAL_HOST and HADR_REMOTE_HOST configuration parameters. The database attempts to resolve the two parameters to the same IP format. The following table shows how the IP mode is determined for IPv6-enabled servers:

IP mode used for HADR_LOCAL_HOST	IP mode used for HADR_REMOTE_HOST	IP mode used for HADR communications
IPv4 address	IPv4 address	IPv4
IPv4 address	IPv6 address	Error
IPv4 address	hostname, maps to v4 only	IPv4
IPv4 address	hostname, maps to v6 only	Error
IPv4 address	hostname, maps to v4 and v6	IPv4
IPv6 address	IPv4 address	Error

IP mode used for HADR_LOCAL_HOST	IP mode used for HADR_REMOTE_HOST	IP mode used for HADR communications
IPv6 address	IPv6 address	IPv6
IPv6 address	hostname, maps to v4 only	Error
IPv6 address	hostname, maps to v6 only	IPv6
IPv6 address	hostname, maps to v4 and v6	IPv6
hostname, maps to v4 only	IPv4 address	IPv4
hostname, maps to v4 only	IPv6 address	Error
hostname, maps to v4 only	hostname, maps to v4 only	IPv4
hostname, maps to v4 only	hostname, maps to v6 only	Error
hostname, maps to v4 only	hostname, maps to v4 and v6	IPv4
hostname, maps to v6 only	IPv4 address	Error
hostname, maps to v6 only	IPv6 address	IPv6
hostname, maps to v6 only	hostname, maps to v4 only	Error
hostname, maps to v6 only	hostname, maps to v6 only	IPv6
hostname, maps to v6 only	hostname, maps to v4 and v6	IPv6
hostname, maps to v4 and v6	IPv4 address	IPv4
hostname, maps to v4 and v6	IPv6 address	IPv6
hostname, maps to v4 and v6	hostname, maps to v4 only	IPv4
hostname, maps to v4 and v6	hostname, maps to v6 only	IPv6
hostname, maps to v4 and v6	hostname, maps to v4 and v6	IPv6

The primary and standby databases can make HADR connections only if they use the same format. If one server is IPv6 enabled (but also supports IPv4) and the other server only supports IPv4, at least one of the HADR_LOCAL_HOST or HADR_REMOTE_HOST parameters must specify an IPv4 address. This tells the database to use IPv4 even if the server supports IPv6.

When you specify values for the high availability disaster recovery (HADR) local service and remote service parameters (HADR_LOCAL_SVC and HADR_REMOTE_SVC) while preparing an update database configuration command, the values you specify must be ports that are not in use for any other service, including other DB2 components or other HADR databases. In particular, you cannot set either parameter value to the TCP/IP port used by the server to await communications from remote clients (the SVCENAME database manager configuration parameter) or the next port (SVCENAME + 1).

If the primary and standby databases are on different machines, they can use the same port number or service name; otherwise, different values should be used. The HADR_LOCAL_SVC and HADR_REMOTE_SVC parameters can be set to either a port number or a service name.

The synchronization mode (HADR_SYNCMODE) and time-out period (HADR_TIMEOUT) must be identical on both the primary and standby databases. The consistency of these configuration parameters is checked when an HADR pair establishes a connection.

TCP connections are used for communication between the primary and standby databases. A primary database that is not connected to a standby database, either because it is starting up or because the connection is lost, will listen on its local port for new connections. A standby database that is not connected to a primary database will continue issuing connection requests to its remote host.

Although the local host and local service parameters (HADR_LOCAL_HOST, HADR_LOCAL_SVC) are only used on the primary database, you should still set them on the standby database to ensure that they are ready if the standby database has to take over as the primary database.

When the primary database starts, it waits for a standby to connect for a minimum of 30 seconds or for the number of seconds specified by the value of the HADR_TIMEOUT database configuration parameter, whichever is longer. If the standby does not connect in the specified time, the startup will fail. (The one exception to this is when the START HADR command is issued with the BY FORCE option.)

After an HADR pair establishes a connection, they will exchange heart beat messages. The heartbeat interval is one-quarter of the value of the HADR_TIMEOUT database configuration parameter, or 30 seconds, whichever is shorter. The HADR_HEARTBEAT monitor element shows the number of heartbeats a database expected to receive but did not receive from the other database. If one database does not receive any message from the other database within the number of seconds specified by HADR_TIMEOUT, it will initiate a disconnect. This means that at most it takes the number of seconds specified by HADR_TIMEOUT for a primary to detect the failure of either the standby or the intervening network. If you set the HADR_TIMEOUT configuration parameter too low, you will receive false alarms and frequent disconnections.

If the HADR_PEER_WINDOW database configuration parameter is set to zero, then when the primary and standby databases are in peer state, problems with the standby or network will only block primary transaction processing for the number of seconds specified by the HADR_TIMEOUT configuration parameter, at most. If you set the HADR_PEER_WINDOW to a non-zero value, then the primary database will not commit transactions until connection with the standby database is restored, or the HADR_PEER_WINDOW time value elapses, whichever happens first.

Note: For maximal availability, the default value for the HADR_PEER_WINDOW database configuration parameter is zero. When HADR_PEER_WINDOW is set to zero, then as soon as the connection between the primary and the standby is closed (either because the standby closed the connection, a network error is detected, or timeout is reached), the primary drops out of peer state to avoid blocking transactions. For increased data consistency, but reduced availability, you can set the HADR_PEER_WINDOW database configuration parameter to a non-zero value, which will cause the primary database to remain in disconnected peer state for the length of time specified by the HADR_PEER_WINDOW value.

The following sample configuration is for the primary and standby databases.

On the primary:

```
   HADR_LOCAL_HOST          host1.ibm.com
   HADR_LOCAL_SVC           hadr_service
   HADR_REMOTE_HOST         host2.ibm.com
   HADR_REMOTE_SVC          hadr_service
```

```
HADR_REMOTE_INST         dbinst2
HADR_TIMEOUT             120
HADR_SYNCMODE            NEARSYNC
HADR_PEER_WINDOW         120
```

On the standby:

```
HADR_LOCAL_HOST          host2.ibm.com
HADR_LOCAL_SVC           hadr_service
HADR_REMOTE_HOST         host1.ibm.com
HADR_REMOTE_SVC          hadr_service
HADR_REMOTE_INST         dbinst1
HADR_TIMEOUT             120
HADR_SYNCMODE            NEARSYNC
HADR_PEER_WINDOW         120
```

Log archiving configuration for DB2 High Availability Disaster Recovery (HADR)

To use log archiving with DB2 High Availability Disaster Recovery (HADR), configure both the primary database and the standby database for automatic log retrieval capability from all log archive locations.

If either the standby database or the primary database is unable to access all log archive locations, then you must manually copy log files from the log archive to the following locations:

- the standby database logpath or archive location for local catchup
- the primary database logpath or archive location for remote catchup

Only the current primary database can perform log archiving. If the primary and standby databases are set up with separate archiving locations, logs are archived only to the primary database's archiving location. In the event of a takeover, the standby database becomes the new primary database and any logs archived from that point on are saved to the original standby database's archiving location. In such a configuration, logs are archived to one location or the other, but not both; with the exception that following a takeover, the new primary database might archive a few logs that the original primary database had already archived.

After a takeover, if the new primary database (original standby database) experiences a media failure and needs to perform a restore and rollforward, it might need to access logs that only exist in the original primary database archive location.

The standby database will not delete a log file from its local logpath until it has been notified by the primary database that the primary database has archived it. This behavior provides added protection against the loss of log files. If the primary database fails and its log disk becomes corrupted before a particular log file is archived on the primary database, the standby database will not delete that log file from its own disk because it has not received notification that the primary database successfully archived the log file. If the standby database then takes over as the new primary database, it will archive that log file before recycling it. If both the *logarchmeth1* and *logarchmeth2* configuration parameters are in use, the standby database will not recycle a log file until the primary database has archived it using both methods.

High availability disaster recovery (HADR) performance

Configuring different aspects of your database system, including network bandwidth, CPU power, and buffer size, can improve the performance of your DB2 High Availability Disaster Recovery (HADR) databases.

For optimum HADR performance, consider the following recommendations for managing your system:

- Network bandwidth must be greater than the database log generation rate.
- Network delays affect the primary only in SYNC and NEARSYNC modes.
- The slowdown in system performance as a result of using SYNC mode can be significantly larger than that of the other synchronization modes. In SYNC mode, the primary database sends log pages to the standby database only after the log pages have been successfully written to the primary database log disk. In order to protect the integrity of the system, the primary database waits for an acknowledgement from the standby before notifying an application that a transaction was prepared or committed. The standby database sends the acknowledgement only after it writes the received log pages to the standby database disk. The resulting overhead is: the log write on the standby database plus round-trip messaging.
- In NEARSYNC mode, the primary database writes and sends log pages in parallel. The primary then waits for an acknowledgement from the standby. The standby database acknowledges as soon as the log pages are received into its memory. On a fast network, the overhead to the primary database is minimal. The acknowledgement might have already arrived by the time the primary database finishes local log write.
- For ASYNC mode, the log write and send are also in parallel; however, in this mode the primary database does not wait for an acknowledgement from the standby. Therefore, network delay is not an issue. Performance overhead is even smaller with ASYNC mode than with NEARSYNC mode.
- For each log write on the primary, the same log pages are also sent to the standby. Each write operation is called a *flush*. The size of the flush is limited to the log buffer size on the primary database (which is controlled by the database configuration parameter *logbufsz*). The exact size of each flush is nondeterministic. A larger log buffer does not necessarily lead to a larger flush size.
- The standby database should be powerful enough to replay the logged operations of the database as fast as they are generated on the primary. Identical primary and standby hardware is recommended.
- In most systems, the logging capability is not driven to its limit. Even in SYNC mode, there might not be an observable slow down on the primary database. For example, if the limit of logging is 40 Mb per second with HADR enabled, but the system was just running at 30 Mb per second before HADR is enabled, then you might not notice any difference in overall system performance.
- To speed up the catchup process, you can use a shared log archive device. However, if the shared device is a serial device such as a tape drive, you might experience performance degradation on both the primary and standby databases because of mixed read and write operations.

Network congestion

If the standby database is too slow replaying log pages, its log-receiving buffer might fill up, thereby preventing the buffer from receiving more log pages. In SYNC and NEARSYNC modes, if the primary database flushes its log buffer one more time, the data will likely be buffered in the network pipeline consisting of the primary machine, the network, and the standby database. Because the standby database does not have free buffer to receive the data, it cannot acknowledge, so the primary database becomes blocked while waiting for the standby database's acknowledgement.

In ASYNC mode, the primary database continues to send log pages until the pipeline fills up and it cannot send additional log pages. This condition is called *congestion*. Congestion is reported by the **hadr_connect_status** monitor element. For SYNC and NEARSYNC modes, the pipeline can usually absorb a single flush and congestion will not occur. However, the primary database remains blocked waiting for an acknowledgement from the standby database on the flush operation.

Congestion can also occur if the standby database is replaying log records that take a long time to replay, such as database or table reorganization log records.

Increasing the size of the standby database log-receiving buffer can help to reduce congestion, although it might not remove all of the causes of congestion. By default, the size of the standby database log-receiving buffer is two times the size of the primary database log-writing buffer. The database configuration parameter *logbufsz* specifies the size of the primary database log-writing buffer. The DB2 registry variable **DB2_HADR_BUF_SIZE** can be used to tune the size of the standby database log-receiving buffer.

Registry variable DB2_HADR_PEER_WAIT_LIMIT

When registry variable **DB2_HADR_PEER_WAIT_LIMIT** is set, the HADR primary database will break out of the peer state if logging on the primary database has been blocked for the specified number of seconds because of log replication to the standby. When this limit is reached, the primary database will break the connection to the standby database. If the peer window is disabled, the primary will enter disconnected state and logging resumes. If the peer window is enabled, the primary database will enter disconnected peer state, in which logging continues to be blocked. The primary database leaves disconnected peer state upon re-connection or peer window expiration. Logging resumes once the primary database leaves disconnected peer state.

Honoring peer window transition when breaking out of peer state ensures peer window semantics for safe takeover in all cases. If the primary fails during the transition, normal peer window protection still applies (safe takeover from standby as long as it's still in disconnected-peer state).

Registry variables DB2_HADR_SOSNDBUF and DB2_HADR_SORCVBUF

To maximize network and HADR performance, the TCP socket buffer sizes may require tuning. HADR log shipping workload, network bandwidth, and transmission delay are important factors to consider when tuning the TCP socket buffer sizes. If you change the TCP socket buffer size at the system level, he settings are applied to all TCP connections on the machine. Setting a large system level socket buffer size will consume a large amount of memory.

Two registry variables, **DB2_HADR_SOSNDBUF** and **DB2_HADR_SORCVBUF** allow tuning of the TCP socket send and receive buffer size for HADR connections only. These two variables have the value range of 1024 to 4294967295 and default to the socket buffer size of the operating system, which will vary depending on the operating system. Some operating systems will automatically round or silently cap the user specified value.

Cluster managers and high availability disaster recovery (HADR)

You can implement DB2 High Availability Disaster Recovery (HADR) databases on nodes of a cluster, and use a cluster manager to improve the availability of your database solution. You can have both the primary database and the standby database managed by the same cluster manager, or you can have the primary database and the standby database managed by different cluster managers.

Set up an HADR pair where the primary and standby databases are serviced by the same cluster manager

This configuration is best suited to environments where the primary and standby databases are located at the same site and where the fastest possible failover is required. These environments would benefit from using HADR to maintain DBMS availability, rather using crash recovery or another recovery method.

You can use the cluster manager to quickly detect a problem and to initiate a takeover operation. Because HADR requires separate storage for the DBMS, the cluster manager should be configured with separate volume control. This configuration prevents the cluster manager from waiting for failover to occur on the volume before using the DBMS on the standby system. You can use the automatic client reroute feature to redirect client applications to the new primary database.

Set up an HADR pair where the primary and standby databases are not serviced by the same cluster manager

This configuration is best suited to environments where the primary and standby databases are located at different sites and where high availability is required for disaster recovery in the event of a complete site failure. There are several ways you can implement this configuration. When an HADR primary or standby database is part of a cluster, there are two possible failover scenarios.

- If a partial site failure occurs and a node to which the DBMS can fail over remains available, you can choose to perform a cluster failover. In this case, the IP address and volume failover is performed using the cluster manager; HADR is not affected.
- If a complete site failure occurs where the primary database is located, you can use HADR to maintain DBMS availability by initiating a takeover operation. If a complete site failure occurs where the standby database is located, you can repair the site or move the standby database to another site.

Initializing a standby database

One strategy for making a database solution highly available is maintaining a primary database to respond to user application requests, and a secondary or standby database that can take over database operations for the primary database if the primary database fails. Initializing the standby database entails copying the primary database to the standby database.

There are several ways to initialize the standby database. For example:

- Use disk mirroring to copy the primary database, and use DB2 database suspended I/O support to split the mirror to create the second database.

- Create a backup image of the primary database and recovery that image to the standby database.
- Use SQL replication to capture data from the primary database and apply that data to the standby database.

After initializing the standby database, you must configure your database solution to synchronize the primary database and standby database so the standby database can take over for the primary database if the primary database fails.

Using a split mirror as a standby database

Use the following procedure to create a split mirror of a database for use as a standby database. If a failure occurs on the primary database and crash recovery is necessary, you can use the standby database to take over for the primary database.

To use a split mirror as a standby database, follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
   While the database is suspended, you should not be running other utilities or tools. You should only be making a copy of the database.
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.

   Note: Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. Catalog the mirrored database on the secondary system.

   Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the db2relocatedb utility or the RELOCATE USING option of the db2inidb command to accomplish this.
5. Start the database instance on the secondary system:
   ```
   db2start
   ```
6. Initialize the mirrored database on the secondary system by placing it in rollforward pending state:
   ```
   db2inidb database_alias as standby
   ```

   If required, specify the RELOCATE USING option of the db2inidb command to relocate the standby database:
   ```
   db2inidb database_alias as standby relocate using relocatedbcfg.txt
   ```

   where the relocatedbcfg.txt file contains the information required to relocate the database.

   Note:

   a. You can take a full database backup using the split mirror if you have DMS table spaces (database managed space) or automatic storage table spaces. Taking a backup using the split mirror off-loads the overhead of taking a backup on the production database.

   b. The database directory (including the volume directory), the log directory, and the container directories must be moved to the desired location before you use the RELOCATE USING option.

7. Set up a user exit program to retrieve the log files from the primary system.
8. Roll the database forward to the end of the logs or to a point-in-time.
9. Continue retrieving log files, and rolling the database forward through the logs until you reach the end of the logs or the point-in-time required for the standby database.
10. To bring the standby database online issue the ROLLFORWARD command with the STOP option specified.

    Note: The logs from the primary database cannot be applied to the mirrored database once it has been taken out of rollforward pending state.

Configuring DB2 High Availability Disaster Recovery (HADR) synchronization mode

The HADR_SYNCMODE configuration parameter determines the degree of protection your DB2 High Availability Disaster Recovery (HADR) database solution has against transaction loss. The synchronization mode determines when the primary database server considers a transaction complete, based on the state of the logging on the standby database. The more strict the synch mode configuration parameter value, the more protection your database solution has against transaction data loss, but the slower your transaction processing performance. You must balance the need for protection against transaction loss with the need for performance.

These modes apply only when the primary and standby databases are in peer or disconnected peer state.

Use the HADR_SYNCMODE configuration parameter to set the synchronization mode. Valid values are:

SYNC (synchronous)

This mode provides the greatest protection against transaction loss, and using it results in the longest transaction response time among the three modes.

In this mode, log writes are considered successful only when logs have been written to log files on the primary database and when the primary database has received acknowledgement from the standby database that the logs have also been written to log files on the standby database. The log data is guaranteed to be stored at both sites.

If the standby database crashes before it can replay the log records, the next time it starts it can retrieve and replay them from its local log files. If the primary database fails, a failover to the standby database guarantees that any transaction that has been committed on the primary database has also been committed on the standby database. After the failover operation, when the client reconnects to the new primary database, there can be transactions committed on the new primary database that were never reported as committed to the original primary. This occurs when the primary database fails before it processes an acknowledgement message from the standby database. Client applications should consider querying the database to determine whether any such transactions exist.

If the primary database loses its connection to the standby database, what happens next depends on the configuration of the hadr_peer_window database configuration parameter. If hadr_peer_window is set to a non-zero time value, then upon losing connection with the standby database the primary database will move into disconnected peer state and continue to wait for acknowledgement from the standby database before committing transactions. If the hadr_peer_window database configuration parameter is set to zero, the primary and standby databases are no longer considered to be in peer state and transactions will not be held back waiting for acknowledgement from the standby database. If the failover operation is performed when the databases are not in peer or disconnected peer state, there is no guarantee that all of the transactions committed on the primary database will appear on the standby database.

If the primary database fails when the databases are in peer or disconnected peer state, it can rejoin the HADR pair as a standby database after a failover operation. Because a transaction is not considered to be committed until the primary database receives acknowledgement from the standby database that the logs have also been written to log files on the standby database, the log sequence on the primary will be the same as the log sequence on the standby database. The original primary database (now a standby database) just needs to catch up by replaying the new log records generated on the new primary database since the failover operation.

If the primary database is not in peer state when it fails, its log sequence might be different from the log sequence on the standby database. If a failover operation has to be performed, the log sequence on the primary and standby databases might be different because the standby database starts its own log sequence after the failover. Because some operations cannot be undone (for example, dropping a table), it is not possible to revert the primary database to the point in time when the new log sequence was created. If the log sequences are different and you issue the START HADR command with the STANDBY option on the original primary, you will receive a message that the command was successful. However, this message is issued before reintegration is attempted. If reintegration fails, pair validation messages will be issued to the administration log and the diagnostics log on both the primary and the standby. The reintegrated standby will remain the standby, but the primary will reject the standby during pair validation causing the standby database to shut down. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

NEARSYNC (near synchronous)

While this mode has a shorter transaction response time than synchronous mode, it also provides slightly less protection against transaction loss.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and when the primary database has received acknowledgement from the standby system that the logs have also been written to main memory on the standby system. Loss of data occurs only if both sites fail simultaneously and if the target site has not transferred to nonvolatile storage all of the log data that it has received.

If the standby database crashes before it can copy the log records from memory to disk, the log records will be lost on the standby database. Usually, the standby database can get the missing log records from the primary database when the standby database restarts. However, if a failure on the primary database or the network makes retrieval impossible and a failover is required, the log records will never appear on the standby database, and transactions associated with these log records will never appear on the standby database.

If transactions are lost, the new primary database is not identical to the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database cannot to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records (because both the primary and standby databases have failed), the log sequences on the primary and standby databases will be different and attempts to restart the original primary database as a standby database without first performing a restore operation will fail. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

ASYNC (asynchronous)

This mode has the highest chance of transaction loss if the primary system fails. It also has the shortest transaction response time among the three modes.

In this mode, log writes are considered successful only when the log records have been written to the log files on the primary database and have been delivered to the TCP layer of the primary system's host machine. Because the primary system does not wait for acknowledgement from the standby system, transactions might be considered committed when they are still on their way to the standby.

A failure on the primary database host machine, on the network, or on the standby database can cause log records in transit to be lost. If the primary database is available, the missing log records can be resent to the standby database when the pair reestablishes a connection. However, if a failover operation is required while there are missing log records, those log records will never reach the standby database, causing the associated transactions to be lost in the failover.

If transactions are lost, the new primary database is not exactly the same as the original primary database after a failover operation. Client applications should consider resubmitting these transactions to bring the application state up to date.

If the primary database fails when the primary and standby databases are in peer state, it is possible that the original primary database will not be able to rejoin the HADR pair as a standby database without being reinitialized using a full restore operation. If the failover involves lost log records, the log sequences on the primary and standby databases will be different, and attempts to restart the original primary database as a standby database will fail. Because there is a greater possibility of log records being lost if a failover occurs in asynchronous mode, there is also a greater possibility that the primary database will not be able to rejoin the HADR pair. If the original primary database successfully rejoins the HADR pair, you can achieve failback of the database by issuing the TAKEOVER HADR command without specifying the BY FORCE option. If the original primary database cannot rejoin the HADR pair, you can reinitialize it as a standby database by restoring a backup image of the new primary database.

High availability disaster recovery (HADR) support

To get the most out of the DB2 database High Availability Disaster Recovery (HADR) feature, consider system requirements and feature limitations when designing your high availability database solution.

System requirements for High Availability Disaster Recovery (HADR)

To achieve optimal performance with High Availability Disaster Recovery (HADR), ensure that your system meets the following requirements for hardware, operating systems, and for the DB2 database system.

Recommendation: For better performance, use the same hardware and software for the system where the primary database resides and for the system where the standby database resides. If the system where the standby database resides has fewer resources than the system where the primary database resides, it is possible that the standby database will be unable to keep up with the transaction load generated by the primary database. This can cause the standby database to fall behind or the performance of the primary database to degrade. In a failover situation, the new primary database should have the resources to service the client applications adequately.

Hardware and operating system requirements

Recommendation: Use identical host computers for the HADR primary and standby databases. That is, they should be from the same vendor and have the same architecture.

The operating system on the primary and standby databases should be the same version, including patches. You can violate this rule for a short time during a rolling upgrade, but take extreme caution.

A TCP/IP interface must be available between the HADR host machines, and a high-speed, high-capacity network is recommended.

DB2 database requirements

The versions of the database systems for the primary and standby databases must be identical; for example, both must be either version 8 or version 9. During rolling upgrades, the modification level (for example, the fix pack level) of the database system for the standby database can be later than that of the primary database for a short while to test the new level. However, you should not keep this configuration for an extended period of time. The primary and standby databases will not connect to each other if the modification level of the database system for the primary database is later than that of the standby database.

The DB2 database software for both the primary and standby databases must have the same bit size (32 or 64 bit). Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

If you create a table space on the primary database and log replay fails on the standby database because the containers are not available, the primary database does not receive an error message stating that the log replay failed.

To check for log replay errors, you must monitor the db2diag log and the administration notification log on the standby database when you are creating new table spaces.

If a takeover operation occurs, the new table space that you created is not available on the new primary database. To recover from this situation, restore the table space on the new primary database from a backup image.

In the following example, table space MY_TABLESPACE is restored on database MY_DATABASE before it is used as the new primary database:

1. `db2 connect to my_database`
2. `db2 list tablespaces show detail`

   Note: Run the db2 list tablespaces show detail command to show the status of all table spaces and to obtain the table space ID number required for Step 5.
3. `db2 stop hadr on database my_database`
4. `db2 "restore database my_database tablespace (my_tablespace) online redirect"`
5. `db2 "set tablespace containers for my_tablespace_ID_# ignore rollforward container operations using (path '/my_new_container_path/')"`
6. `db2 "restore database my_database continue"`
7. `db2 rollforward database my_database to end of logs and stop tablespace "(my_tablespace)"`
8. `db2 start hadr on database my_database as primary`

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases.

Automatic storage databases are fully supported by HADR, including replication of the ALTER DATABASE statement with the ADD STORAGE ON clause. Similar to table space containers, the storage path must exist on both primary and standby.

The primary and standby databases must have the same database name. This means that they must be in different instances.

Redirected restore is not supported. That is, HADR does not support redirecting table space containers. However, database directory and log directory changes are supported. Table space containers created by relative paths will be restored to paths relative to the new database directory.

Buffer pool requirements

Since buffer pool operations are also replayed on the standby database, it is important that the primary and standby databases have the same amount of memory.

Installation and storage requirements for High Availability Disaster Recovery (HADR)

To achieve optimal performance with High Availability Disaster Recovery (HADR), ensure that your system meets the following installation and storage requirements.

Installation requirements

For HADR, instance paths should be the same on the primary and the standby databases. Using different instance paths can cause problems in some situations, such as if an SQL stored procedure invokes a user-defined function (UDF) and the path to the UDF object code is expected to be on the same directory for both the primary and standby server.

Storage requirements

Automatic storage databases are fully supported by HADR, including replication of the ALTER DATABASE statement with the ADD STORAGE ON clause. Similar to table space containers, the storage path must exist on both primary and standby. Symbolic links can be used to create identical paths. The primary and standby databases can be on the same computer. Even though their database storage starts at the same path, they do not conflict because the actual directories used have instance names embedded in them (since the primary and standby databases must have the same database name, they must be in different instances). The storage path is formulated as storage_path_name/inst_name/dbpart_name/db_name/tbsp_name/container_name.

Table spaces and their containers must be identical on the primary and standby databases. Properties that must be identical include: the table space type (DMS or SMS), table space size, container path, container size, and container file type (raw device or file system). If the database is enabled for automatic storage then the storage paths must be identical. This includes the path names and the amount of space on each that is devoted to the database. The amount of space allocated for log files should also be the same on both the primary and standby databases.

When you issue a table space statement on the primary database, such as CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE, it is replayed on the standby database. You must ensure that the devices involved are set up on both of the databases before you issue the table space statement on the primary database.

If the table space setup is not identical on the primary and standby databases, log replay on the standby database might encounter errors such as OUT OF SPACE or TABLE SPACE CONTAINER NOT FOUND. Similarly, if the databases are enabled for automatic storage and the storage paths are not identical, log records associated with the ADD STORAGE ON clause of the ALTER DATABASE statement will not be replayed. As a result, the existing storage paths might prematurely run out of space on the standby system and automatic storage table spaces will not be able to increase in size. If any of these situations occurs, the affected table space is put in rollforward pending state and is ignored in subsequent log replay. If a takeover operation occurs, the table space will not be available to applications.

If the problem is noticed on the standby system prior to a takeover then the resolution is to re-establish the standby database while addressing the storage issues. The steps to do this include:
- Deactivating the standby database.
- Dropping the standby database.
- Ensuring the necessary filesystems exist with enough free space for the subsequent restore and rollforward.
- Restoring the database at the standby system using a recent backup of the primary database (or, reinitialize using split mirror or flash copy with the db2inidb command). If the primary database is enabled for automatic storage then do not redefine the storage paths during the restore. Also, table space containers should not be redirected as part of the restore.
- Restarting HADR on the standby system.

However, if the problem is noticed with the standby database after a takeover has occurred (or if a choice was made to not address the storage issues until this time) then the resolution is based on the type of problem that was encountered.

If the database is enabled for automatic storage and space is not available on the storage paths associated with the standby database then follow these steps:
1. Make space available on the storage paths by extending the filesystems, or by removing unnecessary non-DB2 files on them.
2. Perform a table space rollforward to the end of logs.

In the case where the addition or extension of containers as part of log replay could not occur, if the necessary backup images and log file archives are available, you might be able to recover the table space by first issuing the SET TABLESPACE CONTAINERS statement with the IGNORE ROLLFORWARD CONTAINER OPERATIONS option and then issuing the ROLLFORWARD command.

The primary and standby databases do not require the same database path. If relative container paths are used, the same relative path might map to different absolute container paths on the primary and standby databases. Consequently, if the primary and standby databases are placed on the same computer, all table space containers must be defined with relative paths so that they map to different paths for primary and standby.

HADR and Network Address Translation (NAT) support

NAT helps alleviate the inevitable shortage of available IPv4 addresses. It is seen as a quick-fix alternative to the IPv6 solution (which may take decades to implement). NAT is supported in an HADR environment.

HADR and NAT support

HADR always cross checks local and remote host configurations on the primary and standby nodes.

In a NAT environment, a host is known to itself by one IP address, but known to the other hosts by a different IP address. This behavior will cause the HADR host cross check to fail.

To avoid this situation in a NAT environment, you can set registry variable **DB2_HADR_NO_IP_CHECK** to ON. This will bypass the host cross-check, enabling the primary and standby to connect in a NAT environment.

It is recommended to leave the **DB2_HADR_NO_IP_CHECK** registry variable OFF, the default setting, if you are not running in a NAT environment. Disabling the cross check weakens HADR's validation of your configuration.

Restrictions for high Availability Disaster Recovery (HADR)

To achieve optimal performance with High Availability Disaster Recovery (HADR), consider HADR restrictions when designing your high availability DB2 database solution.

The following list is a summary of High Availability Disaster Recovery (HADR) restrictions:

- HADR is not supported in a partitioned database environment.
- The primary and standby databases must have the same operating system version and the same version of the DB2 database system, except for a short time during a rolling upgrade.
- The DB2 database system software on the primary and standby databases must be the same bit size (32 or 64 bit).
- Reads on the standby database are not supported. Clients cannot connect to the standby database.
- Log archiving can only be performed by the current primary database.
- Self Tuning Memory Manager (STMM) can be run only on the current primary database. After the primary database is started or the standby database is converted to a primary database by takeover, the STMM EDU may not start until the first client connection comes in.
- Backup operations are not supported on the standby database.
- Non-logged operations, such as changes to database configuration parameters and to the recovery history file, are not replicated to the standby database.
- Load operations with the COPY NO option specified are not supported.
- HADR does not support the use of raw I/O (direct disk access) for database log files. If HADR is started via the START HADR command, or the database is activated (restarted) with HADR configured, and raw logs are detected, the associated command will fail.
- For one-phase commit, an HADR database can act as either a federated server (Transaction Manager), or a data source (Resource Manager). For two-phase commit, an HADR database can only act as the data source.

Scheduling maintenance for high availability

Your DB2 database solution will require regular maintenance. You will have to perform maintenance such as: software or hardware upgrades; database performance tuning; database backups; statistics collection and monitoring for business purposes. You must minimize the impact of these maintenance activities on the availability of your database solution.

Before you can schedule maintenance activities, you must identify those maintenance activities that you will have to perform on your database solution.

To schedule maintenance, perform the following steps:

1. Identify periods of low database activity.

   It is best to schedule maintenance activities for low-usage times (those periods of time when the fewest user applications are making requests of the database system.) Depending on the type of business applications you are creating, there might even be periods of time when no user applications are accessing the database system.
2. Categorize the maintenance activities you must perform according to the following:
   - The maintenance can be automated
   - You must bring the database solution offline while you perform the maintenance
   - You can perform the maintenance while the database solution is online
3. For those maintenance activities that can be automated, configure automated maintenance using one of the following methods:
   - Use the auto_maint configuration parameter
   - Use the Configure Automatic Maintenance Wizard
   - Use one of the system stored procedure called AUTOMAINT_SET_POLICY and AUTOMAINT_SET_POLICYFILE
4. If any of the maintenance activities you must perform require the database server to be offline, schedule those offline maintenance activities for those low-usage times.
5. For those maintenance activities that can be performed while the database server is online:
   - Identify the availability impact of running those online maintenance activities.
   - Schedule those online maintenance activities so as to minimize the impact of running those maintenance activities on the availability of the database system.

   For example: schedule online maintenance activities for low-usage times; and use throttling mechanisms to balance the amount of system resources the maintenance activities use.

Collecting automated maintenance policy information using SYSPROC.AUTOMAINT_GET_POLICY or SYSPROC.AUTOMAINT_GET_POLICYFILE

You can use the system stored procedures AUTOMAINT_GET_POLICY and AUTOMAINT_GET_POLICYFILE to retrieve the automated maintenance policy configured for a database.

To retrieve the automated maintenance policy for a database, perform the following steps:

1. Connect to the database
2. Call AUTOMAINT_GET_POLICY or AUTOMAINT_GET_POLICYFILE
   - The parameters required for AUTOMAINT_GET_POLICY are:
     a. Maintenance type, specifying the type of automated maintenance activity about which to return information.

   b. Pointer to a BLOB in which the procedure will return the automated maintenance policy information in XML format.
- The parameters required for AUTOMAINT_GET_POLICYFILE are:
   a. Maintenance type, specifying the type of automated maintenance activity about which to return information.
   b. The name of a file to which the procedure will print the automated maintenance policy information.

Valid maintenance type values are:
- AUTO_BACKUP - automatic backup
- AUTO_REORG - automatic table and index reorganization
- AUTO_RUNSTATS - automatic table runstats operations
- MAINTENANCE_WINDOW - maintenance window

Configuring an automated maintenance policy using SYSPROC.AUTOMAINT_SET_POLICY or SYSPROC.AUTOMAINT_SET_POLICYFILE

You can use the system stored procedures AUTOMAINT_SET_POLICY and AUTOMAINT_SET_POLICYFILE to configure the automated maintenance policy for a database.

To configure the automated maintenance policy for a database, perform the following steps:
1. Connect to the database
2. Call AUTOMAINT_SET_POLICY or AUTOMAINT_SET_POLICYFILE
   - The parameters required for AUTOMAINT_SET_POLICY are:
     a. Maintenance type, specifying the type of automated maintenance activity to configure.
     b. Pointer to a BLOB that specifies the automated maintenance policy in XML format.
   - The parameters required for AUTOMAINT_SET_POLICYFILE are:
     a. Maintenance type, specifying the type of automated maintenance activity to configure.
     b. The name of an XML file that specifies the automated maintenance policy.

   Valid maintenance type values are:
   - AUTO_BACKUP - automatic backup
   - AUTO_REORG - automatic table and index reorganization
   - AUTO_RUNSTATS - automatic table runstats operations
   - MAINTENANCE_WINDOW - maintenance window

Sample automated maintenance policy specification XML for AUTOMAINT_SET_POLICY or AUTOMAINT_SET_POLICYFILE

Whether you are using AUTOMAINT_SET_POLICY or AUTOMAINT_SET_POLICYFILE to specify your automated maintenance policy, you must specify the policy using XML. There are sample files in SQLLIB/samples/automaintcfg that demonstrate how to specify your automated maintenance policy in XML.

The second parameter you pass to the system stored procedure AUTOMAINT_SET_POLICY is a BLOB containing XML, specifying your desired automated maintenance policy. The second parameter you pass to the system stored procedure AUTOMAINT_SET_POLICYFILE is the name of an XML file that specifies your desired automated maintenance policy. The XML elements that are valid in the BLOB you pass to AUTOMAINT_SET_POLICY are the same elements that are valid in the XML file you pass to AUTOMAINT_SET_POLICYFILE.

In the samples directory SQLLIB/samples/automaintcfg there are four XML files that contain example automated maintenance policy specification:

DB2MaintenanceWindowPolicySample.xml

demonstrates specifying a maintenance window during which time the database manager should schedule automated maintenance.

DB2AutoBackupPolicySample.xml

demonstrates specifying how the database manager should perform automatic backup.

DB2AutoReorgPolicySample.xml

demonstrates specifying how the database manager should perform automatic table and index reorganization (including reclaiming extents from multidimensional clustering (MDC) tables).

DB2DefaultAutoRunstatsPolicySample.xml

demonstrates specifying how the database manager should perform automatic table runstats operations.

You can create your own automated maintenance policy specification XML by copying the XML from these files and modifying that XML according to the requirements of your system.

Configuring database logging options

Use database logging configuration parameters to specify data logging options for your database, such as the type of logging to use, the size of the log files, and the location where log files should be stored.

To configure database logging options, you must have SYSADM, SYSCTRL, or SYSMAINT authority.

You can configure database logging options using the UPDATE DATABASE CONFIGURATION command on the command line processor (CLP), through the Configure Database Logging wizard GUI in the Control Center, or by calling the db2CfgSet API.

- To configure database logging options using the UPDATE DATABASE CONFIGURATION command on the command line processor:
  1. Specify whether you want to use circular logging or archive logging. If you want to use circular logging, the LOGARCHMETH1 and LOGARCHMETH2 database configuration parameters must be set to OFF. This is the default setting. To use archive logging, you must set at least one of these database configuration parameters to a value other than OFF. For example, if you want to use archive logging and you want to save the archived logs to disk, issue the following:

     ```
     db2 update db configuration for mydb using logarchmeth1
        disk:/u/dbuser/archived_logs
     ```

The archived logs will be placed in a directory called /u/dbuser/archived_logs.

2. Specify values for other database logging configuration parameters, as required. The following are additional configuration parameters for database logging:
   - ARCHRETRYDELAY
   - BLK_LOG_DSK_FUL
   - FAILARCHPATH
   - LOGARCHOPT1
   - LOGARCHOPT2
   - LOGBUFSZ
   - LOGFILSIZ
   - LOGPRIMARY
   - LOGRETAIN
   - LOGSECOND
   - MAX_LOG
   - MIRRORLOGPATH
   - NEWLOGPATH
   - MINCOMMIT
   - NUMARCHRETRY
   - NUM_LOG_SPAN
   - OVERFLOWLOGPATH
   - USEREXIT

   For more information on these database logging configuration parameters, refer to "Configuration parameters for database logging."

- To open the Configure Database Logging wizard:
  1. From the Control Center, expand the object tree until you find the database for which you want to set up logging.
  2. Right-click on the database and select Configure Database Logging from the pop-up menu. The Configure Database Logging wizard opens.
- Detailed information is provided through the online help facility within the Control Center.

Configuration parameters for database logging

A key element of any high availability strategy is database logging. You can use database logs to record transaction information, synchronize primary and secondary or standby databases, and rollforward a secondary database that has taken over for a failed primary database. To configure these database logging activities according to your needs, you must set a variety of database configuration parameters.

Archive Retry Delay (archretrydelay)
: Specifies the amount of time (in seconds) to wait between attempts to archive log files after the previous attempt fails. The default value is 20.

Block on log disk full (blk_log_dsk_ful)
: This configuration parameter can be set to prevent disk full errors from being generated when DB2 cannot create a new log file in the active log path. Instead, DB2 will attempt to create the log file every five minutes until it succeeds. After each attempt, DB2 will write a message to the

administration notification log. The only way to confirm that your application is hanging because of a log disk full condition is to monitor the administration notification log. Until the log file is successfully created, any user application that attempts to update table data will not be able to commit transactions. Read-only queries might not be directly affected; however, if a query needs to access data that is locked by an update request or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Setting *blk_log_dsk_ful* to `YES` causes applications to hang when DB2 encounters a log disk full error. You are then able to resolve the error and the transaction can continue. A disk full situation can be resolved by moving old log files to another file system, or by increasing the size of the file system so that hanging applications can complete.

If blk_log_dsk_ful is set to `NO`, a transaction that receives a log disk full error will fail and be rolled back. In some cases, the database will come down if a transaction causes a log disk full error.

Failover Archive Path (failarchpath)

Specifies an alternate directory for the archive log files if the log archive method specified fails. This directory is a temporary storage area for the log files until the log archive method that failed becomes available again at which time the log files will be moved from this directory to the log archive method. By moving the log files to this temporary location, log directory full situations might be avoided. This parameter must be a fully qualified existing directory.

Log archive method 1 (logarchmeth1), log archive method 2 (logarchmeth2)

These parameters cause the database manager to archive log files to a location that is not the active log path. If both of these parameters are specified, each log file is archived twice. This means that you will have two copies of archived log files in two different locations.

Valid values for these parameters include a media type and, in some cases, a target field. Use a colon (:) to separate the values. Valid values are:

OFF Specifies that the log archiving method is not to be used. If both *logarchmeth1* and *logarchmeth2* are set to `OFF`, the database is considered to be using circular logging and will not be rollforward recoverable. This is the default.

LOGRETAIN

This value can only be used for *logarchmeth1* and is equivalent to setting the *logretain* configuration parameter to `RECOVERY`. If you specify this value, the *logretain* configuration parameters will automatically be updated.

USEREXIT

This value is only valid for *logarchmeth1* and is equivalent to setting the *userexit* configuration parameter to `ON`. If specify this value, the *userexit* configuration parameter will be automatically updated.

DISK This value must be followed by a colon(:) and then a fully qualified existing path name where the log files will be archived. For example, if you set *logarchmeth1* to `DISK:/u/dbuser/archived_logs` the archive log files will be placed in a directory called `/u/dbuser/archived_logs`.

Note: If you are archiving to tape, you can use the db2tapemgr utility to store and retrieve log files.

TSM If specified without any additional configuration parameters, this value indicates that log files should be archived on the local TSM server using the default management class. If followed by a colon(:) and a TSM management class, the log files will be archived using the specified management class.

VENDOR
Specifies that a vendor library will be used to archive the log files. This value must be followed by a colon(:) and the name of the library. The APIs provided in the library must use the backup and restore APIs for vendor products.

Note:

1. If either *logarchmeth1* or *logarchmeth2* is set to a value other than `OFF`, the database is configured for rollforward recovery.
2. If you update the *userexit* or *logretain* configuration parameters *logarchmeth1* will automatically be updated and vice versa. However, if you are using either *userexit* or *logretain*, *logarchmeth2* must be set to `OFF`.

Log archive options 1 (logarchopt1), log archive options 2 (logarchopt2)
Specifies a string which is passed on to the TSM server or vendor APIs. For TSM, this field is used to allow the database to retrieve logs that were generated on a different TSM node or by a different TSM user. The string must be provided in the following format:

```
"-fromnode=nodename -fromowner=ownername"
```

where `nodename` is the name of the TSM node that originally archived the log files, and `ownername` is the name of the TSM user that originally archived the log files. Each log archive options field corresponds to one of the log archive methods: *logarchopt1* is used with *logarchmeth1*, and *logarchopt2* is used with *logarchmeth2*.

Log Buffer (logbufsz)
This parameter allows you to specify the amount of memory to use as a buffer for log records before writing these records to disk. The log records are written to disk when any one of the following events occurs:

- A transaction commits
- The log buffer becomes full
- Some other internal database manager event occurs.

Increasing the log buffer size results in more efficient input/output (I/O) activity associated with logging, because the log records are written to disk less frequently, and more records are written each time. However, recovery can take longer with a larger log buffer size value.

Log file size (logfilsiz)
This parameter specifies the size of each configured log, in number of 4-KB pages.

There is a 1024 GB logical limit on the total active log space that you can configure. This limit is the result of the upper limit for each log file, which is 4 GB, and the maximum combined number of primary and secondary log files, which is 256.

The size of the log file has a direct bearing on performance. There is a performance cost for switching from one log to another. So, from a pure performance perspective, the larger the log file size the better. This parameter also indicates the log file size for archiving. In this case, a larger log file is size it not necessarily better, since a larger log file size can increase the chance of failure or cause a delay in log shipping scenarios. When considering active log space, it might be better to have a larger number of smaller log files. For example, if there are 2 very large log files and a transaction starts close to the end of one log file, only half of the log space remains available.

Every time a database is deactivated (all connections to the database are terminated), the log file that is currently being written is truncated. So, if a database is frequently being deactivated, it is better not to choose a large log file size because DB2 will create a large file only to have it truncated. You can use the ACTIVATE DATABASE command to avoid this cost, and having the buffer pool primed will also help with performance.

Assuming that you have an application that keeps the database open to minimize processing time when opening the database, the log file size should be determined by the amount of time it takes to make offline archived log copies.

Minimizing log file loss is also an important consideration when setting the log size. Archiving takes an entire log. If you use a single large log, you increase the time between archiving. If the medium containing the log fails, some transaction information will probably be lost. Decreasing the log size increases the frequency of archiving but can reduce the amount of information loss in case of a media failure since the smaller logs before the one lost can be used.

Log retain (logretain)

This configuration parameter has been replaced by *logarchmeth1*. It is still supported for compatibility with previous versions of DB2.

If logretain is set to `RECOVERY`, archived logs are kept in the database log path directory, and the database is considered to be recoverable, meaning that rollforward recovery is enabled.

Note: The default value for the *logretain* database configuration parameter does not support rollforward recovery. You must change the value of this parameter if you are going to use rollforward recovery.

Maximum log per transaction (max_log)

This parameter indicates the percentage of primary log space that can be consumed by one transaction. The value is a percentage of the value specified for the logprimary configuration parameter.

If the value is set to `0`, there is no limit to the percentage of total primary log space that a transaction can consume. If an application violates the *max_log* configuration, the application will be forced to disconnect from the database, the transaction will be rolled back, and error SQL1224N will be returned.

You can override this behavior by setting the DB2_FORCE_APP_ON_MAX_LOG registry variable to *FALSE*. This will cause transactions that violate the *max_log* configuration to fail and return error SQL0964N. The application can still commit the work completed by previous statements in the unit or work, or it can roll the work completed back to undo the unit of work.

This parameter, along with the num_log_span configuration parameter, can be useful when infinite active logspace is enabled. If infinite logging is on (that is, if logsecond is -1) then transactions are not restricted to the upper limit of the number of log files (logprimary + logsecond). When the value of logprimary is reached, DB2 starts to archive the active logs, rather than failing the transaction. This can cause problems if, for instance, there is a long running transactions that has been left uncommitted (perhaps caused by a bad application). If this occurs, the active logspace keeps growing, which might lead to poor crash recovery performance. To prevent this, you can specify values for either one or both of the max_log or num_log_span configuration parameters.

Note: The following DB2 commands are excluded from the limitation imposed by the *max_log* configuration parameter: ARCHIVE LOG, BACKUP DATABASE, LOAD, REORG, RESTORE DATABASE, and ROLLFORWARD DATABASE.

Mirror log path (mirrorlogpath)

To protect the logs on the primary log path from disk failure or accidental deletion, you can specify that an identical set of logs be maintained on a secondary (mirror) log path. To do this, change the value of this configuration parameter to point to a different directory. Active logs that are currently stored in the mirrored log path directory are not moved to the new location if the database is configured for rollforward recovery.

Because you can change the log path location, the logs needed for rollforward recovery might exist in different directories. You can change the value of this configuration parameter during a rollforward operation to allow you to access logs in multiple locations.

You must keep track of the location of the logs.

Changes are not applied until the database is in a consistent state. The configuration parameter *database_consistent* returns the status of the database.

To turn this configuration parameter off, set its value to DEFAULT.

Note:

1. This configuration parameter is not supported if the primary log path is a raw device.
2. The value specified for this parameter cannot be a raw device.

New log path (newlogpath)

The database logs are initially created in SQLOGDIR, which is a subdirectory of the database directory. You can change the location in which active logs and future archived logs are placed by changing the value of this configuration parameter to point to a different directory or to a device. Active logs that are currently stored in the database log path directory are not moved to the new location if the database is configured for rollforward recovery.

Because you can change the log path location, the logs needed for rollforward recovery might exist in different directories or on different devices. You can change the value of this configuration parameter during a rollforward operation to allow you to access logs in multiple locations.

You must keep track of the location of the logs.

Changes are not applied until the database is in a consistent state. The configuration parameter *database_consistent* returns the status of the database.

Number of Commits to Group (mincommit)

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed. This delay can help reduce the database manager overhead associated with writing log records and, as a result, improve performance when you have multiple applications running against a database, and many commits are requested by the applications within a very short period of time.

The grouping of commits occurs only if the value of this parameter is greater than 1, and if the number of applications connected to the database is greater than the value of this parameter. When commit grouping is in effect, application commit requests are held until either one second has elapsed, or the number of commit requests equals the value of this parameter.

Number of archive retries on error (numarchretry)

Specifies the number of attempts that will be made to archive log files using the specified log archive method before they are archived to the path specified by the *failarchpath* configuration parameter. This parameter can only be used if the *failarchpath* configuration parameter is set. The default value is 5.

Number log span (num_log_span)

This parameter indicates the number of active log files that an active transaction can span. If the value is set to 0, there is no limit to how many log files one single transaction can span.

If an application violates the *num_log_span* configuration, the application will be forced to disconnect from the database and error SQL1224N will be returned.

This parameter, along with the max_log configuration parameter, can be useful when infinite active logspace is enabled. If infinite logging is on (that is, if logsecond is -1) then transactions are not restricted to the upper limit of the number of log files (logprimary + logsecond). When the value of logprimary is reached, DB2 starts to archive the active logs, rather than failing the transaction. This can cause problems if, for instance, there is a long running transactions that has been left uncommitted (perhaps caused by a bad application). If this occurs, the active logspace keeps growing, which might lead to poor crash recovery performance. To prevent this, you can specify values for either one or both of the max_log or num_log_span configuration parameters.

Note: The following DB2 commands are excluded from the limitation imposed by the *num_log_span* configuration parameter: ARCHIVE LOG, BACKUP DATABASE, LOAD, REORG, RESTORE DATABASE, and ROLLFORWARD DATABASE.

Overflow log path (overflowlogpath)

This parameter can be used for several functions, depending on your logging requirements. You can specify a location for DB2 to find log files that are needed for a rollforward operation. It is similar to the OVERFLOW LOG PATH option of the ROLLFORWARD command; however, instead of specifying the OVERFLOW LOG PATH option for every ROLLFORWARD command issued, you can set this configuration parameter once. If both are used, the OVERFLOW LOG PATH option will overwrite the *overflowlogpath* configuration parameter for that rollforward operation.

If *logsecond* is set to -1, you can specify a directory for DB2 to store active log files retrieved from the archive. (Active log files must be retrieved for rollback operations if they are no longer in the active log path).

If *overflowlogpath* is not specified, DB2 will retrieve the log files into the active log path. By specifying this parameter you can provide additional resource for DB2 to store the retrieved log files. The benefit includes spreading the I/O cost to different disks, and allowing more log files to be stored in the active log path.

For example, if you are using the db2ReadLog API for replication, you can use *overflowlogpath* to specify a location for DB2 to search for log files that are needed for this API. If the log file is not found (in either the active log path or the overflow log path) and the database is configured with *userexit* enabled, DB2 will retrieve the log file. You can also use this parameter to specify a directory for DB2 to store the retrieved log files. The benefit comes from reducing the I/O cost on the active log path and allowing more log files to be stored in the active log path.

If you have configured a raw device for the active log path, *overflowlogpath* must be configured if you want to set *logsecond* to -1, or if you want to use the db2ReadLog API.

To set *overflowlogpath*, specify a string of up to 242 bytes. The string must point to a path name, and it must be a fully qualified path name, not a relative path name. The path name must be a directory, not a raw device.

Note: In a partitioned database environment, the database partition number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

Primary logs (logprimary)

This parameter specifies the number of primary logs of size *logfilsiz* that will be created.

A primary log, whether empty or full, requires the same amount of disk space. Thus, if you configure more logs than you need, you use disk space unnecessarily. If you configure too few logs, you can encounter a log-full condition. As you select the number of logs to configure, you must consider the size you make each log and whether your application can handle a log-full condition. The total log file size limit on active log space is 256 GB.

If you are enabling an existing database for rollforward recovery, change the number of primary logs to the sum of the number of primary and secondary logs, plus 1. Additional information is logged for LONG VARCHAR and LOB fields in a database enabled for rollforward recovery.

Secondary logs (logsecond)

This parameter specifies the number of secondary log files that are created and used for recovery, if needed.

If the primary log files become full, secondary log files (of size *logfilsiz*) are allocated, one at a time as needed, up to the maximum number specified by this parameter. If this parameter is set to -1, the database is configured with infinite active log space. There is no limit on the size or number of in-flight transactions running on the database. Infinite active logging is useful in environments that must accommodate large jobs requiring more log space than you would normally allocate to the primary logs.

Note:

1. Log archiving must be enabled in order to set *logsecond* to -1.
2. If this parameter is set to -1, crash recovery time might be increased since DB2 might need to retrieve archived log files.

User exit (userexit)

This configuration parameter has been replaced by *logarchmeth1*. It is still supported for compatibility with previous versions of DB2.

This parameter causes the database manager to call a user exit program for archiving and retrieving logs. The log files are archived in a location that is different from the active log path. If *userexit* is set to `ON`, rollforward recovery is enabled.

The data transfer speed of the device you use to store offline archived logs, and the software used to make the copies, must at a minimum match the average rate at which the database manager writes data in the logs. If the transfer speed cannot keep up with new log data being generated, you might run out of disk space if logging activity continues for a sufficiently long period of time. The amount of time it takes to run out of disk space is determined by the amount of free disk space. If this happens, database processing stops.

The data transfer speed is most significant when using tape or an optical medium. Some tape devices require the same amount of time to copy a file, regardless of its size. You must determine the capabilities of your archiving device.

Tape devices have other considerations. The frequency of the archiving request is important. For example, if the time taken to complete any copy operation is five minutes, the log should be large enough to hold five minutes of log data during your peak work load. The tape device might have design limits that restrict the number of operations per day. These factors must be considered when you determine the log size.

Note:

1. This value must be set to `ON` to enable infinite active log space.
2. The default value for the *userexit* database configuration parameter does not support rollforward recovery, and must be changed if you are going to use it.

Reducing logging with the NOT LOGGED INITIALLY parameter

If your application creates and populates work tables from master tables, and you are not concerned about the recoverability of these work tables because they can be easily recreated from the master tables, you can create the work tables specifying the NOT LOGGED INITIALLY parameter on the CREATE TABLE statement. This reduces logging and improves performance.

The advantage of using the NOT LOGGED INITIALLY parameter is that any changes made on a table (including insert, delete, update, or create index operations) in the same unit of work that creates the table will not be logged. This not only reduces the logging that is done, but can also increase the performance of your application. You can achieve the same result for existing tables by using the ALTER TABLE statement with the NOT LOGGED INITIALLY parameter.

Note:

1. You can create more than one table with the NOT LOGGED INITIALLY parameter in the same unit of work.
2. Changes to the catalog tables and other user tables are still logged.

Because changes to the table are not logged, you should consider the following when deciding to use the NOT LOGGED INITIALLY table attribute:

- *All* changes to the table will be flushed out to disk at commit time. This means that the commit might take longer.
- If the NOT LOGGED INITIALLY attribute is activated and an activity occurs that is not logged, the entire unit of work will be rolled back if a statement fails or a ROLLBACK TO SAVEPOINT is executed (SQL1476N).
- If you are using high availability disaster recovery (HADR) you should not use the NOT LOGGED INITIALLY table attribute. Tables created on the primary database with the NOT LOGGED INITIALLY option specified are not replicated to the standby database. Attempts to access such tables after an HADR standby database takes over as the primary database will result in an error.
- You cannot recover these tables when rolling forward. If the rollforward operation encounters a table that was created or altered with the NOT LOGGED INITIALLY option, the table is marked as unavailable. After the database is recovered, any attempt to access the table returns SQL1477N.

  Note: When a table is created, row locks are held on the catalog tables until a COMMIT is done. To take advantage of the no logging behavior, you must populate the table in the same unit of work in which it is created. This has implications for concurrency.

Reducing logging with declared temporary tables

If you plan to use declared temporary tables as work tables, note the following:

- Declared temporary tables are not created in the catalogs; therefore locks are not held.
- Logging is not performed against declared temporary tables, even after the first COMMIT.
- Use the ON COMMIT PRESERVE option to keep the rows in the table after a COMMIT; otherwise, all rows will be deleted.
- Only the application that creates the declared temporary table can access that instance of the table.
- The table is implicitly dropped when the application connection to the database is dropped.
- Errors in operation during a unit of work using a declared temporary table do not cause the unit of work to be completely rolled back. However, an error in operation in a statement changing the contents of a declared temporary table will delete all the rows in that table. A rollback of the unit of work (or a savepoint) will delete all rows in declared temporary tables that were modified in that unit of work (or savepoint).

Blocking transactions when the log directory is full

When the DB2 database manager cannot create a new log file in the active log path because there is not enough room for the new file, you will get errors indicating the disk is full. If you set the blk_log_dsk_ful database configuration parameter, the DB2 database manager will repeatedly attempt to create the new log file until the file is successfully created instead of returning “disk full” errors.

If you set the blk_log_dsk_ful database configuration parameter, the DB2 database manager attempts to create the log file every five minutes until it succeeds. If a log archiving method is specified, the DB2 database manager also checks for the completion of log file archiving. If an archived log file is archived successfully, the DB2 database manager can rename the inactive log file to the new log file name and continue. After each attempt, the DB2 database manager writes a message to the administration notification log. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the administration notification log.

Until the log file is successfully created, any user application that attempts to update table data will not able to commit transactions. Read-only queries might not be directly affected; however, if a query needs to access data that is locked by an update request, or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Log file management through log archiving

DB2 Data Server log file archiving is complicated by a variety of operating system file handling and scheduling problems. For example: under certain conditions, the DB2 database manager could try to retrieve a log file while the file is being archived; also, if a disk fails just as the DB2 database manager is archiving a queue of log files, those log files (and the transaction data they contain) could be lost. Correctly configuring database logging can prevent these kinds of potential problems from undermining your availability and recovery strategy.

The following are general considerations that apply to all methods of log archiving:

- Specifying a value for the database configuration parameter *logarchmeth1* indicates that you want the database manager to archive files or to retrieve log files during rollforward recovery of databases using the method specified. A request to retrieve a log file is made when the rollforward utility needs a log file that is not found in the log path directory.
- Locally attached tape drives should not be used to store log files if you are using any of the following:
  - infinite logging
  - online table space level recovery
  - replication
  - the Asynchronous Read Log API (db2ReadLog)
  - high availability disaster recovery (HADR)

  Any of these events can cause a log file to be retrieved, which can conflict with log archiving operations.
- If you are using log archiving, the log manager will attempt to archive active logs as they are filled. In some cases, if a database is deactivated before the log manager is able to record the archive as successful, the log manager might try to archive the log again when the database is activated. Thus, a log file can be archived more than once.
- When archiving, a log file is passed to the log manager when it is full, even if the log file is still active and is needed for normal processing. This allows copies of the data to be moved away from volatile media as quickly as possible. The

log file passed to the log manager is retained in the log path directory until it is no longer needed for normal processing. At this point, the disk space is reused.

- When a log file has been archived and it contains no open transactions, the DB2 database manager does not delete the file but renames it as the next log file when such a file is needed. This results in a performance gain, because creating a new log file (instead of renaming the file) causes all pages to be written out to guarantee the disk space. It is more efficient to reuse than to free up and then reacquire the necessary pages on disk.
- The DB2 database manager will *not* retrieve log files during crash recovery or rollback unless the *logsecond* database configuration parameter is set to -1.
- Configuring log archiving does not guarantee rollforward recovery to the point of failure, but only attempts to make the failure window smaller. As log files fill, the log manager will asynchronously archive the logs. Should the disk containing the log fail before a log file is filled, the data in that log file is lost. Also, since the files are queued for archiving, the disk can fail before all the files are copied, causing any log files in the queue to be lost.

  In the case of a failure of the disk or device on which the log path resides, you can use the MIRRORLOGPATH database configuration parameter to ensure that your logs are written to the secondary path, as long as the disk or device on which the mirror log path is located has not also failed.
- The configured size of each individual log file has a direct bearing on log archiving. If each log file is very large, a large amount of data can be lost if a disk fails. If you configure your database to use small log files the log manager will archive the logs more frequently.

  However, if you are moving the data to a slower device such as tape, you might want to have larger log files to prevent the queue from building up. Using larger log files is also recommended if archiving each file requires substantial overhead, such as rewinding the tape device or establishing a connection to the archive media.
- If you are using log archiving, the log manager will attempt to archive active logs as they are filled. In some cases, the log manager will archive a log before it is full. This will occur if the log file is truncated either due to database deactivation, issuing of the ARCHIVE LOG command, at the end of an online backup, or issuing the SET WRITE SUSPEND command.

  Note: To free unused log space, the log file is truncated before it is archived.
- If you are archiving logs and backup images to a tape drive as a storage device for logs and backup images, you need to ensure that the destination for the backup images and the archived logs is not the same tape drive. Since some log archiving can take place while a backup operation is in progress, an error can occur when the two processes are trying to write to the same tape drive at the same time.

The following considerations apply to calling a user exit program or a vendor program for archiving and retrieving log files:

- The DB2 database manager opens a log file in read mode when it starts a user exit program to archive the file. On some platforms, this prevents the user exit program from being able to delete the log file. Other platforms, like AIX, allow processes, including the user exit program, to delete log files. A user exit program should never delete a log file after it is archived, because the file could still be active and needed for crash recovery. The DB2 database manager manages disk space reuse when log files are archived.

- If a user exit or vendor program receives a request to archive a file that does not exist (because there were multiple requests to archive and the file was deleted after the first successful archiving operation), or to retrieve a file that does not exist (because it is located in another directory or the end of the logs has been reached), it should ignore this request and pass a successful return code.
- On Windows operating systems, you cannot use a REXX™ user exit to archive logs.
- The user exit or vendor program should allow for the existence of different log files with the same name after a point in time recovery; it should be written to preserve both log files and to associate those log files with the correct recovery path.
- If a user exit or vendor program is enabled for two or more databases that are using the same tape device to archive log files, and a rollforward operation is taking place on one of the databases, no other database should be active. If another database tries to archive a log file while the rollforward operation is in progress, the logs required for the rollforward operation might not be found or the new log file archived to the tape device might overwrite the log files previously stored on that tape device.

  To prevent either situation from occurring, you can ensure that no other databases on the database partition that calls the user exit program are open during the rollforward operation, or you can write a user exit program to handle this situation.

Configuring a Clustered environment for high availability

Creating a cluster of machines, and using cluster managing software to balance work load on those machines is one strategy for designing a highly available solution. If you install IBM Data Server on one or several of the machines in a cluster, you must configure the cluster manager to properly react to failures that affect the database or databases. Also, you must configure the database manager instances to work properly in the clustered environment.

About this task

Configuring and administering the database instances and the cluster manager manually is complex, time-consuming, and prone to error. The DB2 High Availability (HA) Feature provides infrastructure for enabling the database manager to communicate with your cluster manager when instance configuration changes, such as stopping a database manager instance, require cluster changes.

Procedure

1. Install cluster managing software.

   SA MP is integrated with DB2 Enterprise Server Edition, DB2 Workgroup Server Edition, DB2 Connect Enterprise Server Edition and DB2 Connect Application Server Edition on AIX, Linux, and Solaris SPARC operating systems. It is also integrated with DB2 Express-C Fixed Term License (FTL) and the DB2 High Availability Feature for Express™ Edition on Linux operating systems. On Windows operating systems, the SA MP is bundled with all of these DB2 database products and features, but it is not integrated with the DB2 installer.
2. Configure IBM Data Server database manager instances for your cluster manager, and configure your cluster manager for IBM Data Server.

   DB2 High Availability Instance Configuration Utility (db2haicu) is a text based utility that you can use to configure and administer your highly available databases in a clustered environment. db2haicu collects information about your database instance, your cluster environment, and your cluster manager by querying your system. You supply more information through parameters to the db2haicu call, an input file, or at runtime by providing information at db2haicu prompts.

3. Over time, as your database needs change and you need to modify your database configuration within the clustered environment, continue to keep the database manager instance configuration and the cluster manager configuration synchronized.

   The DB2 High Availability (HA) Feature provides infrastructure for enabling the database manager to communicate with your cluster manager when instance configuration changes, such as stopping a database manager instance, require cluster changes.

   Whether you use db2haicu with SA MP, or you use another cluster manager that supports the DB2 cluster manager API, administering you clustered environment with the DB2 HA Feature is easier than maintaining the database manager configuration and the cluster configuration separately.

Cluster manager integration with the DB2 High Availability (HA) Feature

The DB2 High Availability (HA) Feature enables integration between IBM Data Server and cluster managing software.

When you stop a database manager instance in a clustered environment, you must make your cluster manager aware that the instance is stopped. If the cluster manager is not aware that the instance is stopped, the cluster manager might attempt an operation such as failover on the stopped instance. The DB2 High Availability (HA) Feature provides infrastructure for enabling the database manager to communicate with your cluster manager when instance configuration changes, such as stopping a database manager instance, require cluster changes.

The DB2 HA Feature is composed of the following elements:

- IBM Tivoli System Automation for Multiplatforms (SA MP) is bundled with IBM Data Server on AIX and Linux as part of the DB2 High Availability (HA) Feature, and integrated with the DB2 installer. You can install, upgrade, or uninstall SA MP using either the DB2 installer or the installSAM and uninstallSAM scripts that are included in the IBM Data Server install media.
- In a clustered environment, some database manager instance configuration and administration operations require related cluster configuration changes. The DB2 High Availability (HA) Feature enables the database manager to automatically request cluster manager configuration changes whenever you perform certain database manager instance configuration and administration operations. See: "Configuring a cluster automatically with the DB2 High Availability (HA) Feature" on page 70
- DB2 High Availability Instance Configuration Utility (db2haicu) is a text based utility that you can use to configure and administer your highly available databases in a clustered environment. db2haicu collects information about your database instance, your cluster environment, and your cluster manager by querying your system. You supply more information through parameters to the db2haicu call, an input file, or at runtime by providing information at db2haicu prompts. See: "DB2 High Availability Instance Configuration Utility (db2haicu)" on page 78

- The DB2 cluster manager API defines a set of functions that enable the database manager to communicate configuration changes to the cluster manager. See: "DB2 cluster manager API" on page 113

IBM Tivoli System Automation for Multiplatforms (SA MP)

IBM Tivoli System Automation for Multiplatforms (SA MP) provides high availability and disaster recovery capabilities for AIX, Linux, Solaris SPARC, and Windows.

SA MP is integrated with DB2 Enterprise Server Edition, DB2 Workgroup Server Edition, DB2 Connect Enterprise Server Edition and DB2 Connect Application Server Edition on AIX, Linux, and Solaris SPARC operating systems. It is also integrated with DB2 Express-C Fixed Term License (FTL) and the DB2 High Availability Feature for Express Edition on Linux operating systems. On Windows operating systems, the SA MP is bundled with all of these DB2 database products and features, but it is not integrated with the DB2 installer.

You can use this copy of SA MP to manage the high availability of your DB2 database system; you cannot use it to manage other things in your cluster without buying an upgrade for the SA MP license.

SA MP is the default cluster manager in an IBM Data Server clustered environment on AIX and Linux.

For more information about SA MP, see: http://publib.boulder.ibm.com/tividd/td/IBMTivoliSystemAutomationforMultiplatforms2.2.html. The list of supported operating systems is also available on the following Web site: http://www.ibm.com/software/tivoli/products/sys-auto-linux/platforms.html.

Configuring a cluster automatically with the DB2 High Availability (HA) Feature

In a clustered environment, some database manager instance configuration and administration operations require related cluster configuration changes. The DB2 High Availability (HA) Feature enables the database manager to automatically request cluster manager configuration changes whenever you perform certain database manager instance configuration and administration operations.

Before you begin

To enable the database manager to perform required cluster configuration for database administration tasks, you must *configure the instance for high availability* by using db2haicu to create a *cluster domain* for the instance. For more information, see: "Configuring a clustered environment using DB2 High Availability Instance Configuration Utility (db2haicu)" on page 71.

Procedure

When you perform the following database manager instance configuration and administration operations, the database manager automatically performs related cluster manager configuration for you:
- Starting a database using START DATABASE or db2start
- Stopping a database using STOP DATABASE or db2stop
- Creating a database using CREATE DATABASE
- Adding storage using CREATE TABLESPACE

- Removing storage using ALTER TABLESPACE DROP or DROP TABLESPACE
- Adding or removing storage paths using ALTER DATABASE
- Dropping a database using DROP TABLESPACE
- Restoring a database using the RESTORE DATABASE or db2Restore
- Specifying the table space containers for redirected restore using SET TABLESPACE CONTAINERS
- Rolling a database forward using ROLLFORWARD DATABASE or db2Rollforward
- Recovering a database using RECOVER DATABASE or db2Recover
- Creating event monitors using CREATE EVENT MONITOR
- Dropping event monitors using DROP EVENT MONITOR
- Creating and altering external routines using:
  - CREATE PROCEDURE
  - CREATE FUNCTION
  - CREATE FUNCTION
  - CREATE METHOD
  - ALTER PROCEDURE
  - ALTER FUNCTION
  - ALTER METHOD
- Dropping external routines using:
  - DROP PROCEDURE
  - DROP FUNCTION
  - DROP METHOD
- Start DB2 High Availability Disaster Recovery (HADR) operations for a database using START HADR
- Stop HADR operations for a database using STOP HADR
- Cause an HADR standby database to take over as an HADR primary database using TAKEOVER HADR
- Setting the database manager configuration parameter **DIAGPATH** or **SPM_LOG_PATH**
- Setting the database configuration parameter **NEWLOGPATH**, **OVERFLOWLOGPATH**, **MIRRORLOGPATH**, or **FAILARCHPATH**
- Dropping a database manager instance using db2idrop

Results

When the database manager coordinates the cluster configuration changes for database administration tasks listed, you do not have to perform separate cluster manager operations.

Configuring a clustered environment using DB2 High Availability Instance Configuration Utility (db2haicu)

You can configure and administer your databases in a clustered environment using DB2 High Availability Instance Configuration Utility (db2haicu). When you specify database manager instance configuration details to db2haicu, db2haicu communicates the required cluster configuration details to your cluster managing software.

Before you begin

- To use DB2 High Availability with IBM Tivoli System Automation for Multiplatforms (SA MP) Version 3.1 on SUSE Linux Enterprise Server (SLES) 11, prior to running the db2haicu tool to create your HA environment, you must download and install SA MP Version 3.1 Fix Pack 4. To download the required fix pack, see: http://www.ibm.com/software/tivoli/support/sys-auto-multi
- There is a set of tasks you must perform before using DB2 High Availability Instance Configuration Utility (db2haicu). For more information, see: "DB2 High Availability Instance Configuration Utility (db2haicu) prerequisites" on page 109.

Restrictions

There are some restrictions for using DB2 High Availability Instance Configuration Utility (db2haicu). For more information, see: "DB2 High Availability Instance Configuration Utility (db2haicu) restrictions" on page 112.

About this task

You can run db2haicu interactively, or using an XML input file:

Interactive mode
: When you invoke DB2 High Availability Instance Configuration Utility (db2haicu) by running the db2haicu command without specifying an XML input file with the **-f** parameter, the utility runs in interactive mode. In interactive mode, db2haicu displays information and queries you for information in a text-based format. For more information, see: "Running DB2 High Availability Instance Configuration Utility (db2haicu) in interactive mode" on page 80

Batch mode with an XML input file
: You can use the `-f` *`<input-file-name>`* parameter with the db2haicu command to run DB2 High Availability Instance Configuration Utility (db2haicu) with an XML input file specifying your configuration details. Running db2haicu with an XML input file is useful when you must perform configuration tasks multiple times, such as when you have multiple database partitions to be configured for high availability. For more information, see: "Running DB2 High Availability Instance Configuration Utility (db2haicu) with an XML input file" on page 81

Procedure

Perform the following steps for each database manager instance.

1. Create a new cluster domain

   When you run DB2 High Availability Instance Configuration Utility (db2haicu) for the first time for a database manager instance, db2haicu creates a model of your cluster, called a *cluster domain*. For more information, see: "Creating a cluster domain using DB2 High Availability Instance Configuration Utility (db2haicu)" on page 110
2. Continue to refine the cluster domain configuration, and administer and maintain the cluster domain

   When you are modifying the cluster domain model of your clustered environment using db2haicu, the database manager propagates the related changes to your database manager instance and cluster configuration. For more information, see: "Maintaining a cluster domain using DB2 High Availability Instance Configuration Utility (db2haicu)" on page 111

What to do next

DB2 High Availability Instance Configuration Utility (db2haicu) does not have a separate diagnostic log. You can investigate and diagnose db2haicu errors using the database manager diagnostic log, db2diag log file, and the db2pd tool. For more information, see: “Troubleshooting DB2 High Availability Instance Configuration Utility (db2haicu)” on page 112

Cluster domain

A cluster domain is a model that contains information about your cluster elements such databases, mount points, and failover policies. You create a cluster domain using DB2 High Availability Instance Configuration Utility (db2haicu). db2haicu uses the information in the cluster domain to enable configuration and maintenance cluster administration tasks. Also, as part of the DB2 High Availability (HA) Feature, the database manager uses the information in the cluster domain to perform automated cluster administration tasks.

If you add a cluster element to the cluster domain, then that element will be included in any subsequent db2haicu configuration operations, or any automated cluster administration operations that are performed by the database manager as part of the DB2 HA Feature. If you remove a cluster element from the cluster domain, then that element will no longer be included in db2haicu operations or database manager automated cluster administration operations. db2haicu and the database manager can only coordinate with your cluster manager for cluster elements that are in the cluster domain that you create using db2haicu.

You can use db2haicu to create and configure the following cluster domain elements:

- Computers or machines (in a cluster domain context, these are referred to as *cluster domain nodes*)
- Network interface cards or NICs (referred to in db2haicu as *network interfaces*, *interfaces*, *network adaptors*, or *adaptors*)
- IP addresses
- Databases, including High Availability Disaster Recovery (HADR) primary and standby database pairs
- Database partitions
- Mount points and paths, including those paths that are not critical to failover in the event of a failure
- Failover policies
- Quorum devices

Cluster management software:

Cluster management software maximizes the work that a cluster of computers can perform. A cluster manager balances workload to reduce bottlenecks, monitors the health of the elements of the cluster, and manages failover when an element fails. A cluster manager can also help a system administrator to perform administration tasks on elements in the cluster (by rerouting workload off of a computer that needs to be serviced, for example.)

Elements of a cluster

To function properly, the cluster manager must be aware of many details related to the elements of the cluster, and the cluster manager must be aware of the relationships between the elements of the cluster.

Here are some examples of cluster elements of which the cluster manager must be aware:

- Physical or virtual computers, machines, or devices in the cluster (in a cluster context, these are referred to as *cluster nodes*)
- Networks that connect the cluster nodes
- Network interfaces cards that connect the cluster nodes to the networks
- IP addresses of cluster nodes
- Virtual or services IP addresses

Here are some examples of relationships of which the cluster manager must be aware:

- Pairs of cluster nodes that have the same software installed and can failover for one another
- Networks that have the same properties and can be used to failover for one another
- The cluster node to which a virtual IP address is currently associated

Adding or modifying elements of your cluster

To make the cluster manager aware of the elements of your cluster and the relationships between those elements, a system administrator must register the elements with the cluster manager. If a system administrator makes a change to the elements of the cluster, the administrator must communicate that change to the cluster manager. Cluster managers have interfaces to help with these tasks.

Cluster administration is challenging because there is an enormous variety of possible cluster elements. An administrator must be an expert in the hardware and operating systems of the cluster nodes, networking protocols and configuration, and the software installed on the cluster nodes such as database software. Registering the elements of the cluster with the cluster management software, or updating the cluster manager after a system change, can be complex and time consuming.

Using db2haicu to add or modify elements of your cluster

In a DB2 database solution, you can use the DB2 High Availability Instance Configuration Utility (db2haicu) to register the elements of your cluster with the cluster manager, and to update the cluster manager after making an administrative change to your cluster. Using db2haicu simplifies these tasks because once you know the model that db2haicu uses to encapsulate the elements of your cluster and the relationships between those elements, you do not need to be an expert in the idiosyncrasies of your hardware, operating systems, and cluster manager interface to perform the tasks.

Resources and resource groups:

A *resource* is any cluster element such a cluster node, database, mount point, or network interface card that has been registered with a cluster manager. If an element is not registered with the cluster manager, then the cluster manager will not be aware of that element and the cluster manager will not include that element in cluster managing operations. A *resource group* is a logical collection of resources. The resource group is a very powerful construct because relationships and constraints can be defined on resource groups that simplify performing complex administration tasks on the resources in those groups.

When a cluster manager collects resources into groups, the cluster manager can operate on all those resources collectively. For example, if two databases called `database-1` and `database-2` belong to the resource group called `resource-group-A`, then if the cluster manager performs a start operation on `resource-group-A` then both `database-1` and `database-2` would be started by that one cluster manager operation.

Restrictions

- A resource group cannot contain an *equivalency* and an equivalency cannot contain a resource group (An *equivalency* is a set of resources that provide the same functionality as each other and can fail over for each other.)
- A resource can only be in one resource group
- A resource cannot be in a resource group and in an equivalency
- A resource group can contain other resource groups, but the maximum nesting level is 50
- The maximum number or resources that you can collect in a resource group is 100

Quorum devices:

A *quorum device* helps a cluster manager make cluster management decisions when the cluster manager's normal decision process does not produce a clear choice. When a cluster manager has to choose between multiple potential actions, the cluster manager counts how many cluster domain nodes support each of the potential actions; and then cluster manager chooses the action that is supported by the majority of cluster domain nodes. If exactly the same number of cluster domain nodes supports more than one choice, then the cluster manager refers to a quorum device to make the choice.

db2haicu supports the quorum devices listed in the following table.

Table 2. Types of quorum device supported by db2haicu

Quorum device	Description
network	A network quorum device is an IP address to which every cluster domain node can connect at all times.

Networks in a cluster domain:

To configure elements of your cluster domain that are related to networks, you can use DB2 High Availability Instance Configuration Utility (db2haicu) to add a *physical network* to your cluster domain. A physical network is composed of: network interface cards, IP addresses, and subnetwork masks.

Network interface cards

A *network interface card* (NIC) is hardware that connects a computer (also called a *cluster node*) to a network. A NIC is sometimes referred to as an *interface*, a *network adaptor*, or an *adaptor*. When you use db2haicu to add a physical network to your cluster domain, you specify at least one NIC including: the host name of the computer to which the NIC belongs; the name of the NIC on that host computer; and the IP address of the NIC.

IP addresses

An *Internet Protocol address* (IP address) is a unique address on a network. In IP version 4, an IP address is 32 bits large, and is normally expressed in dot-decimal notation like this: `129.30.180.16`. An IP address is composed of a network portion and a host computer portion.

db2haicu does not support IP version 6.

Subnetwork masks

A network can be partitioned into multiple logical subnetworks using *subnetwork masks*. A subnetwork mask is a mechanism for moving some bits of the host portion of an IP address to the network portion of the IP address. When you use db2haicu to add an IP address to your cluster domain, you will sometimes need to specify the subnetwork mask for the IP address. For example, when you use db2haicu to add a NIC, you must specify the subnetwork mask for the IP address of the NIC.

Network equivalencies

An *equivalency* is a set of resources that provide the same functionality as each other and can fail over for each other. When you create a network using db2haicu, the NICs in that network can fail over for each other. Such a network is also refered to an a *network equivalency*.

Network protocols

When you use db2haicu to add a network to your cluster domain, you must specify the type of network protocol being used. Currently, only the TCP/IP network protocol is supported.

Failover policies in a cluster domain:

A *failover policy* specifies how a cluster manager should respond when a cluster element such as a network interface card or a database server fails. In general, a cluster manager will transfer workload away from a failed element to an alternative element that had been previously identified to the cluster manager as an appropriate replacement for the element that failed. This transfer of workload from a failed element to a secondary element is called *failover*.

Round robin failover policy

When you are using a *round robin failover policy*, then if there is a failure associated with one computer in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database manager will restart the work from the failed cluster domain node on any other node that is in the cluster domain.

Mutual failover policy

To configure a *mutual failover policy*, you associate a pair of computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) as a system pair. If there is a failure on one of the nodes in this pair, then the database partitions on the failed node will failover to the other node in the pair. Mutual failover is only available when you have multiple database partitions.

N Plus M failover policy

When you are using a *N Plus M failover policy*, then if there is a failure associated with one computer in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database partitions on the failed node will failover to any other node that is in the cluster domain. N Plus M failover is only available when you have multiple database partitions.

Local restart failover policy

When you use a *local restart failover policy*, then if there is a failure on one of the computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database manager will restart the database in place (or locally) on the same node that failed.

HADR failover policy

When you configure a *HADR failover policy*, you are enabling the DB2 High Availability Disaster Recovery (HADR) feature to manage failover. If an HADR primary database fails, the database manager will move the workload from the failed database to an HADR standby database.

Custom failover policy

When you configure a *custom failover policy*, you create a list of computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) onto which the database manager can failover. If a node in the cluster domain fails, the database manager will move the workload from the failed node to one of the nodes in the list that you specified

Mount points in a cluster domain:

After mounting a file system, you can use DB2 High Availability Instance Configuration Utility (db2haicu) to add that mount point to your cluster domain.

Mount points

On UNIX, Linux, and AIX operating systems, to *mount* a file system means to make that file system available to the operating system. During the mount operation, the operating system performs tasks such as reading index or navigation data structures, and associates a directory path with that mounted file system. That associated directory path that you can use to access the mounted file system is called a *mount point*.

Non-critical mount points or paths

There might be mount points or paths in your cluster that do not need to be failed over in the event of a failure. You can use db2haicu to add a list of those non-critical mount points or paths to your cluster domain. Your cluster manager will not include the mount points or paths in that non-ciritcal list in failover operations.

For example, consider the case where you have a hard drive mounted at `/mnt/driveA` on a computer called `node1` in your cluster. If you decide that it is critical for `/mnt/driveA` to be available, your cluster manager will fail over to keep `/mnt/driveA` available if `node1` fails. However, if you decide that it is acceptable for

`/mnt/driveA` to be unavailable if `node1` fails, then you can indicate to your cluster manager that `/mnt/driveA` is not critical for failover by adding `/mnt/driveA` to the list of non-critical paths. If `/mnt/driveA` is identified as non-critical for failover, then that drive might be unavailable if `node1` fails.

DB2 High Availability Instance Configuration Utility (db2haicu)

DB2 High Availability Instance Configuration Utility (db2haicu) is a text based utility that you can use to configure and administer your highly available databases in a clustered environment. db2haicu collects information about your database instance, your cluster environment, and your cluster manager by querying your system. You supply more information through parameters to the db2haicu call, an input file, or at runtime by providing information at db2haicu prompts.

Syntax

```
db2haicu [ -f <XML-input-file-name> ]
         [ -disable ]
         [ -delete [ dbpartitionnum <db-partition-list> |
                     hadrdb <database-name> ] ]
```

Parameters

The parameters that you pass to the db2haicu command are case-sensitive, and must be in lowercase.

-f <*XML-input-file-name*>
: You can use the **-f** parameter to specify your cluster domain details in an XML input file, `<XML-input-file-name>`. For more information, see: "Running DB2 High Availability Instance Configuration Utility (db2haicu) with an XML input file" on page 81.

-disable
: A database manager instance is considered *configured for high availability* once you have used db2haicu to create a cluster domain for that instance. When a database manager instance is configured for high availability, then whenever you perform certain database manager administrative operations that require related cluster configuration changes, the database manager will communicate those cluster configuration changes to the cluster manager. When the database manager coordinates these cluster management tasks with the cluster manager for you, you do not have to perform a separate cluster manager operation for those administrative tasks. This integration between the database manager and the cluster manager is a function of the DB2 High Availability (HA) Feature.

  You can use the **-disable** parameter to cause a database manager instance to cease to be configured for high availability. If the database manager instance is no longer configured for high availability, then the database manager will not coordinate with the cluster manager if you perform any database manager administrative operations that require related cluster configuration changes.

  To reconfigure a database manager instance for high availability, you can run db2haicu again.

-delete
: You can use the **-delete** parameter to delete resource groups for the current database manager instance.

If you do not use either the **dbpartitionnum** parameter or the **hadrdb** parameter, then db2haicu will remove all the resources groups associated with the current database manager instance.

dbpartitionnum <*db-partition-list*>
: You can use the **dbpartitionnum** parameter to delete resource groups that are associated with the database partitions listed in `<db-partition-list>`. `<db-partition-list>` is a comma-separated list of numbers identifying the database partitions.

hadrdb <*database-name*>
: You can use the **hadrdb** parameter to delete resource groups that are associated with the DB2 High Availability Disaster Recovery (HADR) database named `<database-name>`.

If there are no resource groups left in the cluster domain after db2haicu removes the resource groups, then db2haicu will also remove the cluster domain.

Running db2haicu with the **-delete** parameter causes the current database manager instance to cease to be configured for high availability. If the database manager instance is no longer configured for high availability, then the database manager will not coordinate with the cluster manager if you perform any database manager administrative operations that require related cluster configuration changes.

To reconfigure a database manager instance for high availability, you can run db2haicu again.

DB2 High Availability Instance Configuration Utility (db2haicu) startup mode:

The first time that you run DB2 High Availability Instance Configuration Utility (db2haicu) for a given database manager instance, db2haicu operates in startup mode.

When you run db2haicu, db2haicu examines your database manager instance and your system configuration, and searches for an existing *cluster domain*. A cluster domain is a model that contains information about your cluster elements such databases, mount points, and failover policies. You create a cluster domain using DB2 High Availability Instance Configuration Utility (db2haicu). db2haicu uses the information in the cluster domain to enable configuration and maintenance cluster administration tasks. Also, as part of the DB2 High Availability (HA) Feature, the database manager uses the information in the cluster domain to perform automated cluster administration tasks.

When you run db2haicu for a given database manager instance, and there is no cluster domain that is already created and configured for that instance, db2haicu will immediately begin the process of creating and configuring a new cluster domain. db2haicu creates a new cluster domain by prompting you for information such as a name for the new cluster domain and the hostname of the current machine.

If you create a cluster domain, but do not complete the task of configuring the cluster domain, then the next time you run db2haicu, db2haicu will resume the task of configuring the cluster domain.

After you create and configure a cluster domain for a database manager instance, db2haicu will run in maintenance mode.

DB2 High Availability Instance Configuration Utility (db2haicu) maintenance mode:

When you run DB2 High Availability Instance Configuration Utility (db2haicu) and there is already a cluster domain created for the current database manager instance, db2haicu operates in maintenance mode.

When db2haicu is running in maintenance mode, db2haicu presents you with a list of configuration and administration tasks that you can perform.

db2haicu maintenance tasks include adding cluster elements such as databases or cluster nodes to the cluster domain, and removing elements from the cluster domain. db2haicu maintenance tasks also include modifying the details of cluster domain elements such as the failover policy for the database manager instance.

When you run db2haicu in maintenance mode, db2haicu presents you with a list of operations you can perform on the cluster domain:
- Add or remove cluster nodes (machine identified by hostname)
- Add or remove a network interface (network interface card)
- Add or remove database partitions (partitioned database environment only)
- Add or remove a DB2 High Availability Disaster Recovery (HADR) database
- Add or remove a highly available database
- Add or remove a mount point
- Add or remove an IP address
- Add or remove a non-critical path
- Move database partitions and HADR databases for scheduled maintenance
- Change failover policy for the current instance
- Create a new quorum device for the cluster domain
- Destroy the cluster domain

Running DB2 High Availability Instance Configuration Utility (db2haicu) in interactive mode:

When you invoke DB2 High Availability Instance Configuration Utility (db2haicu) by running the db2haicu command without specifying an XML input file with the **-f** parameter, the utility runs in interactive mode. In interactive mode, db2haicu displays information and queries you for information in a text-based format.

Before you begin
- To use DB2 High Availability with IBM Tivoli System Automation for Multiplatforms (SA MP) Version 3.1 on SUSE Linux Enterprise Server (SLES) 11, prior to running the db2haicu tool to create your HA environment, you must download and install SA MP Version 3.1 Fix Pack 4. To download the required fix pack, see: http://www.ibm.com/software/tivoli/support/sys-auto-multi
- There is a set of tasks you must perform before using DB2 High Availability Instance Configuration Utility (db2haicu). For more information, see: "DB2 High Availability Instance Configuration Utility (db2haicu) prerequisites" on page 109.

About this task

When you run db2haicu in interactive mode, you will see information and questions presented to you in text format on your screen. You can enter the information requested by db2haicu at a prompt at the bottom of your screen.

Procedure

To run db2haicu in interactive mode, call the db2haicu command without the `-f` *<input-file-name>*.

What to do next

DB2 High Availability Instance Configuration Utility (db2haicu) does not have a separate diagnostic log. You can investigate and diagnose db2haicu errors using the database manager diagnostic log, db2diag log file, and the db2pd tool. For more information, see: "Troubleshooting DB2 High Availability Instance Configuration Utility (db2haicu)" on page 112

Running DB2 High Availability Instance Configuration Utility (db2haicu) with an XML input file:

You can use the `-f` *<input-file-name>* parameter with the db2haicu command to run DB2 High Availability Instance Configuration Utility (db2haicu) with an XML input file specifying your configuration details. Running db2haicu with an XML input file is useful when you must perform configuration tasks multiple times, such as when you have multiple database partitions to be configured for high availability.

Before you begin

- To use DB2 High Availability with IBM Tivoli System Automation for Multiplatforms (SA MP) Version 3.1 on SUSE Linux Enterprise Server (SLES) 11, prior to running the db2haicu tool to create your HA environment, you must download and install SA MP Version 3.1 Fix Pack 4. To download the required fix pack, see: http://www.ibm.com/software/tivoli/support/sys-auto-multi
- There is a set of tasks you must perform before using DB2 High Availability Instance Configuration Utility (db2haicu). For more information, see: "DB2 High Availability Instance Configuration Utility (db2haicu) prerequisites" on page 109.

About this task

There is a set of sample XML input files located in the samples subdirectory of the sqllib directory that you can modify and use with db2haicu to configure your clustered environment. For more information, see: "Sample XML input files for DB2 High Availability Instance Configuration Utility (db2haicu)" on page 100

Procedure

1. Create an XML input file.
2. Call db2haicu with the `-f` *<input-file-name>*.

   To configure your clustered environment for the current database manager instance using db2haicu and an input file that you create called db2haicu-input.xml, use the following command:

   ```
   db2haicu -f db2haicu-input.xml
   ```

What to do next

DB2 High Availability Instance Configuration Utility (db2haicu) does not have a separate diagnostic log. You can investigate and diagnose db2haicu errors using the database manager diagnostic log, db2diag log file, and the db2pd tool. For more information, see: "Troubleshooting DB2 High Availability Instance Configuration Utility (db2haicu)" on page 112

DB2 High Availability Instance Configuration Utility (db2haicu) input file XML schema definition:

The DB2 High Availability Instance Configuration Utility (db2haicu) input file XML schema definition (XSD) defines the cluster domain objects that you can specify in a db2haicu XML input file. This db2haicu XSD is located in the file called `db2ha.xsd` in the `sqllib/samples/ha/xml` directory.

`DB2ClusterType`

The root element of the db2haicu XML schema definition (XSD) is `DB2Cluster`, which is of type `DB2ClusterType`. A db2haicu XML input file must begin with a `DB2Cluster` element.

"XML schema definition"
"Subelements"
"Attributes" on page 84
"Usage notes" on page 84

XML schema definition

```
<xs:complexType name='DB2ClusterType'>
  <xs:sequence>
    <xs:element name='DB2ClusterTemplate'
                type='DB2ClusterTemplateType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='ClusterDomain'
                type='ClusterDomainType'
                maxOccurs='unbounded'/>
    <xs:element name='FailoverPolicy'
                type='FailoverPolicyType'
                minOccurs='0'/>
    <xs:element name='DB2PartitionSet'
                type='DB2PartitionSetType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='HADRDBSet'
                type='HADRDBType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='HADBSet'
                type='HADBType'
                minOccurs='0'
                maxOccurs='unbounded'/>
  </xs:sequence>
  <xs:attribute name='clusterManagerName' type='xs:string' use='optional'/>
</xs:complexType>
```

Subelements

DB2ClusterTemplate

Type: `DB2ClusterTemplateType`

Usage notes:
Do not include a `DB2ClusterTemplateType` element in your db2haicu XML input file. The `DB2ClusterTemplateType` element is currently reserved for future use.

ClusterDomain

Type: `ClusterDomainType`

A `ClusterDomainType` element contains specifications about: the machines or computers in the cluster domain (also called *cluster domain nodes*); the *network equivalencies* (groups of networks that can fail over for one another); and the *quorum device* (tie-breaking mechanism).

Occurrence rules:
You must include one or more `ClusterDomain` element in your `DB2ClusterType` element.

FailoverPolicy

Type: `FailoverPolicyType`

A `FailoverPolicyType` element specifies the *failover policy* that the cluster manager should use with the cluster domain.

Occurrence rules:
You can include zero or one `FailoverPolicy` element in your `DB2ClusterType` element.

DB2PartitionSet

Type: `DB2PartitionSetType`

A `DB2PartitionSetType` element contains information about database partitions. The `DB2PartitionSetType` element is only applicable in a partitioned database environment.

Occurrence rules:
You can include zero or more `DB2PartitionSet` elements in your `DB2ClusterType` element, according to the db2haicu db2haicu XML schema definition.

HADRDBSet

Type: `HADRDBType`

A `HADRDBType` element contains a list of High Availability Disaster Recovery (HADR) primary and standby database pairs.

Occurrence rules:
You can include zero or more `HADRDBSet` elements in your `DB2ClusterType` element, according to the db2haicu db2haicu XML schema definition.

Usage notes:

- You must not include `HADRDBSet` in a partitioned database environment.
- If you include `HADRDBSet`, then you must specify a failover policy of `HADRFailover` in the `FailoverPolicy` element.

HADBSet

Type: `HADBType`

A `HADBType` element contains a list of databases to include in the cluster domain, and to make highly available.

Occurrence rules:
You can include zero or more `HADBSet` elements in your `DB2ClusterType` element, according to the db2haicu db2haicu XML schema definition.

Attributes

clusterManagerName (optional)

The `clusterManagerName` attribute specifies the cluster manager.

Valid values for this attribute are specified in the following table:

Table 3. Valid values for the `clusterManager` attribute

`clusterManagerName` value	**Cluster manager product**
TSA	IBM Tivoli System Automation for Multiplatforms (SA MP)

Usage notes

In a single partition database environment, you will usually only create a single cluster domain for each database manager instance.

One possible configuration for a multi-partition database environment is:
- Set the `FailoverPolicy` element to `Mutual`
- In the `DB2Partition` subelement of `DB2PartitionSet`, use the `MutualPair` element to specify two cluster domain nodes that are in a single cluster domain

ClusterDomainType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `ClusterDomainType` element contains specifications about: the machines or computers in the cluster domain (also called *cluster domain nodes*); the *network equivalencies* (groups of networks that can fail over for one another); and the *quorum device* (tie-breaking mechanism).

"Superelements"
"XML schema definition"
"Subelements" on page 85
"Attributes" on page 85

Superelements

The following types of elements contain `ClusterDomainType` subelements:
- `DB2ClusterType`

XML schema definition

```
<xs:complexType name='ClusterDomainType'>
  <xs:sequence>
    <xs:element name='Quorum'
                type='QuorumType'
                minOccurs='0'/>
    <xs:element name='PhysicalNetwork'
                type='PhysicalNetworkType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='ClusterNode'
                type='ClusterNodeType'
                maxOccurs='unbounded'/>
  </xs:sequence>
  <xs:attribute name='domainName'      type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

Quorum

Type: `QuorumType`

A `QuorumType` element specifies the *quorum device* for the cluster domain.

Occurrence rules:
You can include zero or one `Quorum` element in your `ClusterDomainType` element.

PhysicalNetwork

Type: `PhysicalNetworkType`

A `PhysicalNetworkType` element contains network interface cards that can fail over for each other. This kind of network is also called a *network equivalency*.

Occurrence rules:
You can include zero or more `PhysicalNetwork` elements in your `ClusterDomainType` element.

ClusterNode

Type: `ClusterNodeType`

A `ClusterNodeType` element contains information about a particular computer or machine (also called a *cluster domain node*) in the cluster.

Occurrence rules:
You must specify at least one `ClusterNode` element in your `ClusterDomainType` element.

Usage notes
IBM Tivoli System Automation for Multiplatforms (SA MP) supports a maximum of 32 cluster domain nodes. If your cluster manager is SA MP, then you can include a maximum of 32 `ClusterNode` elements in your `ClusterDomainType` element.

Attributes

`domainName` (required)

You must specify a unique name for your `ClusterDomainType` element.

If you are using Reliable Scalable Cluster Technology (RSCT) to manage your cluster, the following restrictions apply to domainName:
- `domainName` can only contain the characters A to Z, a to z, digits 0 to 9, period (.), and underscore (_)
- `domainName` cannot be "IW"

QuorumType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `QuorumType` element specifies the *quorum device* for the cluster domain.

"Superelements" on page 86
"XML schema definition" on page 86
"Subelements" on page 86
"Attributes" on page 86

Superelements

The following types of elements contain `QuorumType` subelements:
- `ClusterDomainType`

XML schema definition

```
<xs:complexType name='QuorumType'>
  <xs:attribute name='quorumDeviceProtocol'
                type='QuorumDeviceProtocolType'
                use='required'/>
  <xs:attribute name='quorumDeviceName'
                type='xs:string'
                use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

quorumDeviceProtocol (required)

`quorumDeviceProtocol` specifies the type of quorum to use.

A *quorum device* helps a cluster manager make cluster management decisions when the cluster manager's normal decision process does not produce a clear choice. When a cluster manager has to choose between multiple potential actions, the cluster manager counts how many cluster domain nodes support each of the potential actions; and then cluster manager chooses the action that is supported by the majority of cluster domain nodes. If exactly the same number of cluster domain nodes supports more than one choice, then the cluster manager refers to a quorum device to make the choice.

The type of the `quorumDeviceProtocol` attribute is `QuorumDeviceProtocolType`.

Here is the XML schema definition for the `QuorumDeviceProtocolType`:

```
<xs:simpleType name='QuorumDeviceProtocolType'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='disk'/>
    <xs:enumeration value='scsi'/>
    <xs:enumeration value='network'/>
    <xs:enumeration value='eckd'/>
    <xs:enumeration value='mns'/>
  </xs:restriction>
</xs:simpleType>
```

Currently supported values for this attribute are specified in the following table:

Table 4. Valid values for the `quorumDeviceProtocol` attribute

`quorumDeviceProtocol` value	**Meaning**
network	A network quorum device is an IP address to which every cluster domain node can connect at all times.

quorumDeviceName (required)

The value of the `quorumDeviceName` depends on the type of quorum device specified in `quorumDeviceProtocol`.

Valid values for this attribute are specified in the following table:

Table 5. Valid values for the `quorumDeviceName` attribute

Value of **`quorumDeviceProtocol`**	Valid value for **`quorumDeviceName`**
network	A string containing a properly formatted IP address. For example: `12.126.4.5` For the IP address that you specify to be valid as a network quarum device, every cluster domain node must be able to access this IP addressed (using the `ping` utility, for example.)

PhysicalNetworkType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `PhysicalNetworkType` element contains network interface cards that can fail over for each other. This kind of network is also called a *network equivalency*.

"Superelements"
"XML schema definition"
"Subelements"
"Attributes" on page 88

Superelements

The following types of elements contain `PhysicalNetworkType` subelements:
- `ClusterDomainType`

XML schema definition

```
<xs:complexType name='PhysicalNetworkType'>
  <xs:sequence>
    <xs:element name='Interface'
                type='InterfaceType'
                minOccurs='1'
                maxOccurs='unbounded'/>
    <xs:element name='LogicalSubnet'
                type='IPAddressType'
                minOccurs='0'
                maxOccurs='unbounded'/>
  </xs:sequence>
  <xs:attribute name='physicalNetworkName'
                type='xs:string'
                use='required'/>
  <xs:attribute name='physicalNetworkProtocol'
                type='PhysicalNetworkProtocolType'
                use='required'/>
</xs:complexType>
```

Subelements

Interface

Type: `InterfaceType`

The `InterfaceType` element consists of an IP address, the name of a computer or machine in the network (also called a *cluster domain node*), and the name of a *network interface card* (NIC) on that cluster domain node.

Occurrence rules:
You must specify one or more `Interface` elements in your `PhysicalNetworkType` element.

LogicalSubnet

Type: `IPAddressType`

A `IPAddressType` element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

Occurrence rules:
You can include zero or more `LogicalSubnet` elements in your `PhysicalNetworkType` element.

Attributes

physicalNetworkName (required)
You must specify a unique `physicalNetworkName` for each `PhysicalNetworkType` element.

physicalNetworkProtocol (required)
The type of the `physicalNetworkProtocol` attribute is `PhysicalNetworkProtocolType`.

Here is the XML schema definition for the `PhysicalNetworkProtocolType` element:

```
<xs:simpleType name='PhysicalNetworkProtocolType'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='ip'/>
    <xs:enumeration value='rs232'/>
    <xs:enumeration value='scsi'/>
    <xs:enumeration value='ssa'/>
    <xs:enumeration value='disk'/>
  </xs:restriction>
</xs:simpleType>
```

Currently supported values for this attribute are specified in the following table:

Table 6. Valid values for the `physicalNetworkProtocol` attribute

`physicalNetworkProtocol` value	Meaning
ip	TCP/IP protocol

InterfaceType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

The `InterfaceType` element consists of an IP address, the name of a computer or machine in the network (also called a *cluster domain node*), and the name of a *network interface card* (NIC) on that cluster domain node.

"Superelements" on page 89
"XML schema definition" on page 89
"Subelements" on page 89
"Attributes" on page 89

Superelements

The following types of elements have `InterfaceType` subelements:
- `PhysicalNetworkType`

XML schema definition

```
<xs:complexType name='InterfaceType'>
  <xs:sequence>
    <xs:element name='IPAddress'       type='IPAddressType'/>
  </xs:sequence>
  <xs:attribute name='interfaceName'   type='xs:string' use='required'/>
  <xs:attribute name='clusterNodeName' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

IPAddress

Type: `IPAddressType`

A `IPAddressType` element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

Occurrence rules:
You must specify exactly one `IPAddress` in your `InterfaceType` element.

Attributes

interfaceName (required)
You must specify the name of a NIC in the `interfaceName` attribute. The NIC that you specify in the `interfaceName` must exist on the cluster domain node that you specify in the `clusterNodeName` attribute.

clusterNodeName (required)
You must specify the name of the cluster domain node that is located at the IP address that you specify in the `IPAddress` element.

IPAddressType XML schema element for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `IPAddressType` element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

“Superelements”
“XML schema definition” on page 90
“Subelements” on page 90
“Attributes” on page 90

Superelements

The following types of elements have `IPAddressType` subelements:
- `PhysicalNetworkType`
- `InterfaceType`
- `DB2PartitionType`

XML schema definition

```
<xs:complexType name='IPAddressType'>
  <xs:attribute name='baseAddress' type='xs:string' use='required'/>
  <xs:attribute name='subnetMask'  type='xs:string' use='required'/>
  <xs:attribute name='networkName' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

baseAddress (required)
: You must specify the base IP address using a string with a valid IP address format: four sets of numbers ranging from 0 to 255, separated by a period. For example:

```
162.148.31.101
```

subnetMask (required)
: You must specify the base IP address using a string with a valid IP address format.

networkName (required)
: You must specify the same value for `networkName` here as you specified for the `physicalNetworkName` attribute of the PhysicalNetworkType element that contains this `IPAddress` element.

ClusterNodeType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `ClusterNodeType` element contains information about a particular computer or machine (also called a *cluster domain node*) in the cluster.

"Superelements"
"XML schema definition"
"Subelements"
"Attributes"

Superelements

The following types of elements have `ClusterNodeType` elements:
- `ClusterDomainType`

XML schema definition

```
<xs:complexType name='ClusterNodeType'>
  <xs:attribute name='clusterNodeName' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

clusterNodeName (required)
: You must specify the name of the cluster domain node.

FailoverPolicyType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `FailoverPolicyType` element specifies the *failover policy* that the cluster manager should use with the cluster domain.

"Superelements"
"XML schema definition"
"Subelements"
"Possible values"

Superelements

The following types of elements contain `InterfaceType` subelements:
- `DB2ClusterType`

XML schema definition

```
<xs:complexType name='FailoverPolicyType'>
  <xs:choice>
    <xs:element name='RoundRobin'
                type='xs:string'
                minOccurs='0' />
    <xs:element name='Mutual'
                type='xs:string'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='NPlusM'
                type='xs:string'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='LocalRestart'
                type='xs:string'
                fixed=''/>
    <xs:element name='HADRFailover'
                type='xs:string'
                fixed=''/>
    <xs:element name='Custom'
                type='xs:string'
                minOccurs='0' />
  </xs:choice>
</xs:complexType>
```

Subelements

None.

Possible values

Select one of the following choices to specify to the cluster manager what type of failover policy to use if there is a failure anywhere in the cluster domain.

A *failover policy* specifies how a cluster manager should respond when a cluster element such as a network interface card or a database server fails. In general, a cluster manager will transfer workload away from a failed element to an alternative element that had been previously identified to the cluster manager as an appropriate replacement for the element that failed. This transfer of workload from a failed element to a secondary element is called *failover*.

RoundRobin
When you are using a *round robin failover policy*, then if there is a failure

associated with one computer in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database manager will restart the work from the failed cluster domain node on any other node that is in the cluster domain.

Mutual
: To configure a *mutual failover policy*, you associate a pair of computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) as a system pair. If there is a failure on one of the nodes in this pair, then the database partitions on the failed node will failover to the other node in the pair. Mutual failover is only available when you have multiple database partitions.

NPlusM
: When you are using a *N Plus M failover policy*, then if there is a failure associated with one computer in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database partitions on the failed node will failover to any other node that is in the cluster domain. N Plus M failover is only available when you have multiple database partitions.

LocalRestart
: When you use a *local restart failover policy*, then if there is a failure on one of the computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) then the database manager will restart the database in place (or locally) on the same node that failed.

HADRFailover
: When you configure a *HADR failover policy*, you are enabling the DB2 High Availability Disaster Recovery (HADR) feature to manage failover. If an HADR primary database fails, the database manager will move the workload from the failed database to an HADR standby database.

Custom
: When you configure a *custom failover policy*, you create a list of computers in the cluster domain (also called *cluster domain nodes* or simply *nodes*) onto which the database manager can failover. If a node in the cluster domain fails, the database manager will move the workload from the failed node to one of the nodes in the list that you specified

DB2PartitionSetType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `DB2PartitionSetType` element contains information about database partitions. The `DB2PartitionSetType` element is only applicable in a partitioned database environment.

"Superelements"
"XML schema definition" on page 93
"Subelements" on page 93
"Attributes" on page 93

Superelements

`InterfaceType` is a subelement of:

- `PhysicalNetworkType`

XML schema definition

```
<xs:complexType name='DB2PartitionSetType'>
  <xs:sequence>
    <xs:element name='DB2Partition'
                type='DB2PartitionType'
                maxOccurs='unbounded'/>
  </xs:sequence>
</xs:complexType>
```

Subelements

DB2Partition

Type: `DB2PartitionType`

A `DB2PartitionType` element specifies a database partition including the DB2 database manager instance to which the database partition belongs and the database partition number.

Occurrence rules:
You must specify one or more `DB2Partition` elements in your `DB2PartitionSetType` element.

Attributes

None.

DB2PartitionType XML schema element for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `DB2PartitionType` element specifies a database partition including the DB2 database manager instance to which the database partition belongs and the database partition number.

"Superelements"
"XML schema definition"
"Subelements" on page 94
"Attributes" on page 94

Superelements

`InterfaceType` is a subelement of:
- `DB2PartitionSetType`

XML schema definition

```
<xs:complexType name='DB2PartitionType'>
  <xs:sequence>
    <xs:element name='VirtualIPAddress'
                type='IPAddressType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='Mount'
                type='MountType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='HADRDB'
                type='HADRDBType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='MutualPair'
                type='MutualPolicyType'
                minOccurs='0'
                maxOccurs='1'/>
    <xs:element name='NPlusMNode'
                type='NPlusMPolicyType'
                minOccurs='0'
                maxOccurs='unbounded'/>
  </xs:sequence>
  <xs:attribute name='instanceName'     type='xs:string'  use='required'/>
  <xs:attribute name='dbpartitionnum'   type='xs:integer' use='required'/>
</xs:complexType>
```

Subelements

VirtualIPAddress
Type: `IPAddressType`

A `IPAddressType` element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

You can omit including `VirtualIPAddress`; or you can include an unbounded number of `VirtualIPAddress` elements in your `DB2PartitionType` element.

Mount
Type: `MountType`

A `MountType` element contains information about a *mount point* such as the file path that identifies the location of the mounted files.

You can omit including `Mount`; or you can include an unbounded number of `Mount` elements in your `DB2PartitionType` element.

HADRDB
Type: `HADRDBType`

A `HADRDBType` element contains a list of High Availability Disaster Recovery (HADR) primary and standby database pairs.

You can omit including `HADRDB`; or you can include an unbounded number of `HADRDB` elements in your `DB2PartitionType` element.

MutualPair
Type: `MutualPolicyType`

A `MutualPolicyType` element contains information about a pair of cluster domain nodes that can failover for each other.

You can omit including `MutualPair`; or you can include exactly one `MutualPair` elements in your `DB2PartitionType` element.

NPlusMNode
Type: `NPlusMPolicyType`

You can omit including `NPlusMNode`; or you can include an unbounded number of `NPlusMNode` elements in your `DB2PartitionType` element.

Attributes

instanceName (required)
In the `instanceName` attribute you must specify the DB2 database manager instance with which this `DB2PartitionType` element is associated.

dbpartitionnum (required)
In the `dbpartitionnum` attribute you must specify the database partition

number that uniquely identifies the database partition (the `dbpartitionnum` number specified in the `db2nodes.cfg` file, for example.)

MountType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `MountType` element contains information about a *mount point* such as the file path that identifies the location of the mounted files.

"Superelements"
"XML schema definition"
"Subelements"
"Attributes"

Superelements

The following types of elements contain `MountType` subelements:
- `DB2PartitionType`

XML schema definition

```
<xs:complexType name='MountType'>
  <xs:attribute name='filesystemPath' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

filesystemPath (required)
Specify the path that was associated with the mount point when the file system was mounted.

MutualPolicyType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `MutualPolicyType` element contains information about a pair of cluster domain nodes that can failover for each other.

"Superelement"
"XML schema definition"
"Subelements" on page 96
"Attributes" on page 96

Superelement

The following types of elements contain `MutualPolicyType` subelements:
- `DB2PartitionType`

XML schema definition

```
<xs:complexType name='MutualPolicyType'>
  <xs:attribute name='systemPairNode1' type='xs:string' use='required'/>
  <xs:attribute name='systemPairNode2' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

systemPairNode1 (required)
In `systemPairNode1` you must specify the name of a cluster domain node that can fail over for the cluster domain node that you specify in `systemPairNode2`.

systemPairNode2 (required)
In `systemPairNode2` you must specify the name of a cluster domain node that can fail over for the cluster domain node that you specify in `systemPairNode1`.

NPlusMPolicyType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

"Superelements"
"XML schema definition"
"Subelements"
"Attributes"

Superelements

The following types of elements contain `NPlusMPolicyType` subelements:
- `DB2PartitionType`

XML schema definition

```
<xs:complexType name='NPlusMPolicyType'>
  <xs:attribute name='standbyNodeName' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

standbyNodeName (required)
In the `standbyNodeName` element, you must specify the name of a cluster domain node to which the partition that contains this `NPlusMPolicyType` element can fail over.

HADRDBType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `HADRDBType` element contains a list of High Availability Disaster Recovery (HADR) primary and standby database pairs.

"Superelements" on page 97
"XML schema definition" on page 97
"Subelements" on page 97
"Attributes" on page 97
"Usage notes" on page 97
"Restrictions" on page 97

Superelements

The following types of elements contain HADRDBType subelements:
- DB2ClusterType
- DB2PartitionType

XML schema definition

```
<xs:complexType name='HADRDBType'>
  <xs:sequence>
    <xs:element name='VirtualIPAddress'
                type='IPAddressType'
                minOccurs='0'
                maxOccurs='unbounded'/>
    <xs:element name='HADRDB'
                type='HADRDBDefn'
                maxOccurs='unbounded'/>
  </xs:sequence>
</xs:complexType>
```

Subelements

VirtualIPAddress

Type: IPAddressType

A IPAddressType element contains all the details of an IP address such as: the *base address*, the *subnet mask*, and the name of the network to which the IP address belongs.

Occurrence rules:
You can including zero or more VirtualIPAddress elements in your HADRDBType element.

HADRDB

Type: HADRDBDefn

A HADRDBDefn element contains information about a High Availability Disaster Recovery (HADR) primary and standby database pair.

Occurrence rules:
You can include one or more VirtualIPAddress elements in your HADRDBType element.

Attributes

None.

Usage notes

If you include a HADRDBType element in the specification for a given cluster domain, then you must also include a FailoverPolicy element specifying HADRFailover in the same cluster domain specification.

Restrictions

You cannot use the HADRDBType element in a partitioned database environment.

HADRDBDefn XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `HADRDBDefn` element contains information about a High Availability Disaster Recovery (HADR) primary and standby database pair.

"Superelements"
"XML schema definition"
"Subelements"
"Attributes"

Superelements

The following types of elements contain `HADRDBDefn` subelements:
- `HADRDBType`

XML schema definition

```
<xs:complexType name='HADRDBDefn'>
  <xs:attribute name='databaseName'      type='xs:string' use='required'/>
  <xs:attribute name='localInstance'     type='xs:string' use='required'/>
  <xs:attribute name='remoteInstance'    type='xs:string' use='required'/>
  <xs:attribute name='localHost'         type='xs:string' use='required'/>
  <xs:attribute name='remoteHost'        type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

databaseName (required)
Enter the name of the HADR database.

localInstance (required)
The `localInstance` is the database manager instance of the HADR primary database.

remoteInstance (required)
The `remoteInstance` is the database manager instance of the HADR standby database.

localHost (required)
The `localHost` is the hostname of the cluster domain node where the HADR primary database is located.

remoteHost (required)
The `remoteHost` is the hostname of the cluster domain node where the HADR standby database is located.

HADBType XML schema definition for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A `HADBType` element contains a list of databases to include in the cluster domain, and to make highly available.

"Superelements" on page 99
"XML schema definition" on page 99
"Subelements" on page 99
"Attributes" on page 99

Superelements

The following types of elements contain HADBType subelements:

- DB2ClusterType

XML schema definition

```
<xs:complexType name='HADBType'>
  <xs:sequence>
    <xs:element name='HADB'         type='HADBDefn' maxOccurs='unbounded'/>
  </xs:sequence>
  <xs:attribute name='instanceName' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

HADB

Type: HADBDefn

A HADBDefn element describes a database to be includede in the cluster domain and made highly available.

Occurrence rules:
You must include one or more HADB elements in your HADBType element.

Attributes

instanceName (required)
In the instanceName attribute, you must specify the DB2 database manager instance to which the databases specified in the HADB elements belong.

HADBDefn XML schema element for DB2 High Availability Instance Configuration Utility (db2haicu) input files:

A HADBDefn element describes a database to be includede in the cluster domain and made highly available.

"Superelements"
"XML schema definition"
"Subelements"
"Attributes" on page 100

Superelements

HADBDefn is a subelement of:

- HADRDBType

XML schema definition

```
<xs:complexType name='HADBDefn'>
  <xs:attribute name='databaseName' type='xs:string' use='required'/>
</xs:complexType>
```

Subelements

None.

Attributes

databaseName (required)
You must specify exactly one database name in the `databaseName` attribute.

Sample XML input files for DB2 High Availability Instance Configuration Utility (db2haicu):

There is a set of sample XML input files located in the samples subdirectory of the sqllib directory that you can modify and use with db2haicu to configure your clustered environment.

db2ha_sample_sharedstorage_mutual.xml:

The sample file `db2ha_sample_sharedstorage_mutual.xml` is an example of an XML input file that you pass to DB2 High Availability Instance Configuration Utility (db2haicu) to specify a new *cluster domain*. `db2ha_sample_sharedstorage_mutual.xml` is located in the `sqllib/samples/ha/xml` directory.

Features

The `db2ha_sample_sharedstorage_mutual.xml` sample domonstrates how to use db2haicu with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): two
- failover policy: mutual
- database partitions: one
- virtual (service) IP addresses: one
- shared mount points for failover: one

XML source

```
<!-- ================================================================= -->
<!-- = Use the DB2 High Availability Instance Configuration Utility  = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify     = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP)      = -->
<!-- = Base Component as the cluster manager.                       = -->
<!-- ================================================================= -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="db2ha.xsd"
           clusterManagerName="TSA"
           version="1.0">

  <!-- ================================================================= -->
  <!-- = Create a cluster domain named db2HAdomain.                     = -->
  <!-- ================================================================= -->
  <ClusterDomain domainName="db2HAdomain">

    <!-- ============================================================== -->
    <!-- = Specify a network quorum device (IP address: 19.126.4.5).   = -->
    <!-- = The IP must be pingable at all times by each of the cluster = -->
    <!-- = domain nodes.                                               = -->
    <!-- ============================================================== -->
    <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

    <!-- ============================================================== -->
    <!-- = Create a network named db2_public_network_0 with an IP      = -->
    <!-- = network protocol.                                           = -->
    <!-- = This network contains two computers: hasys01 and hasys02.   = -->
    <!-- = Each computer has one network interface card (NIC) called   = -->
    <!-- = eth0.                                                       = -->
    <!-- = The IP address of the NIC on hasys01 is 19.126.52.139       = -->
    <!-- = The IP address of the NIC on hasys02 is 19.126.52.140       = -->
    <!-- ============================================================= -->
    <PhysicalNetwork physicalNetworkName="db2_public_network_0"
                     physicalNetworkProtocol="ip">

      <Interface interfaceName="eth0" clusterNodeName="hasys01">
        <IPAddress baseAddress="19.126.52.139"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys02">
        <IPAddress baseAddress="19.126.52.140"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

    </PhysicalNetwork>

    <!-- ============================================================= -->
    <!-- = List the computers (cluster nodes) in the cluster domain.   = -->
    <!-- ============================================================= -->
    <ClusterNode clusterNodeName="hasys01"/>
    <ClusterNode clusterNodeName="hasys02"/>

  </ClusterDomain>


  <!-- =============================================================== -->
  <!-- = The failover policy specifies the order in which the cluster  = -->
  <!-- = domain nodes should fail over.                                = -->
  <!-- =============================================================== -->
  <FailoverPolicy>
     <Mutual />
  </FailoverPolicy>


  <!-- =============================================================== -->
  <!-- = Specify all the details of the database partition             = -->
  <!-- =============================================================== -->
  <DB2PartitionSet>

    <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
       <VirtualIPAddress baseAddress="19.126.52.222"
                         subnetMask="255.255.255.0"
                         networkName="db2_public_network_0"/>
       <Mount filesystemPath="/home/db2inst1"/>
       <MutualPair systemPairNode1="hasys01" systemPairNode2="hasys02" />
    </DB2Partition>

  </DB2PartitionSet>

</DB2Cluster>
```

db2ha_sample_DPF_mutual.xml:

The sample file `db2ha_sample_DPF_mutual.xml` is an example of an XML input file that you pass to DB2 High Availability Instance Configuration Utility (db2haicu) to specify a new *cluster domain*. `db2ha_sample_DPF_mutual.xml` is located in the `sqllib/samples/ha/xml` directory.

Features

The db2ha_sample_DPF_mutual.xml sample domonstrates how to use db2haicu with an XML input file to define a cluster domain with the following details:

- quorum device: network
- computers in the cluster (cluster domain nodes): four
- failover policy: mutual
- database partitions: two
- virtual (service) IP addresses: one
- shared mount points for failover: two
- databases configured for high availability: two

XML source

```
<!-- ================================================================= -->
<!-- = Use the DB2 High Availability Instance Configuration Utility  = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify      = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP)       = -->
<!-- = Base Component as the cluster manager.                        = -->
<!-- ================================================================= -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="db2ha.xsd"
            clusterManagerName="TSA"
            version="1.0">

  <!-- ================================================================= -->
  <!-- = Create a cluster domain named db2HAdomain.                    = -->
  <!-- ================================================================= -->
  <ClusterDomain domainName="db2HAdomain">

    <!-- =============================================================== -->
    <!-- = Specify a network quorum device (IP address: 19.126.4.5).   = -->
    <!-- = The IP must be pingable at all times by each of the cluster = -->
    <!-- = domain nodes.                                               = -->
    <!-- =============================================================== -->
    <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

    <!-- =============================================================== -->
    <!-- = Create a network named db2_public_network_0 with an IP      = -->
    <!-- = network protocol.                                           = -->
    <!-- = This network contains four computers: hasys01, hasys02,     = -->
    <!-- = hasys03, and hasys04.                                       = -->
    <!-- = Each computer has a network interface card called eth0.     = -->
    <!-- = The IP address of eth0 on hasys01 is 19.126.124.30          = -->
    <!-- = The IP address of eth0 on hasys02 is 19.126.124.31          = -->
    <!-- = The IP address of eth0 on hasys03 is 19.126.124.32          = -->
    <!-- = The IP address of eth0 on hasys04 is 19.126.124.33          = -->
    <!-- =============================================================== -->
    <PhysicalNetwork physicalNetworkName="db2_public_network_0"
                     physicalNetworkProtocol="ip">

      <Interface interfaceName="eth0" clusterNodeName="hasys01">
        <IPAddress baseAddress="19.126.124.30"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys02">
        <IPAddress baseAddress="19.126.124.31"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>
    <Interface interfaceName="eth0" clusterNodeName="hasys03">
      <IPAddress baseAddress="19.126.124.32"
                 subnetMask="255.255.255.0"
                 networkName="db2_public_network_0"/>
    </Interface>

    <Interface interfaceName="eth0" clusterNodeName="hasys04">
      <IPAddress baseAddress="19.126.124.33"
                 subnetMask="255.255.255.0"
                 networkName="db2_public_network_0"/>
    </Interface>

  </PhysicalNetwork>

  <!-- =============================================================== -->
  <!-- = Create a network named db2_private_network_0 with an IP     = -->
  <!-- = network protocol.                                           = -->
  <!-- = This network contains four computers: hasys01, hasys02,     = -->
  <!-- = hasys03, and hasys04 (same as db2_public_network_0.)        = -->
  <!-- = In addition to eth0, each computer has a network interface  = -->
  <!-- = card called eth1.                                           = -->
  <!-- = The IP address of eth1 on hasys01 is 192.168.23.101         = -->
  <!-- = The IP address of eth1 on hasys02 is 192.168.23.102         = -->
  <!-- = The IP address of eth1 on hasys03 is 192.168.23.103         = -->
  <!-- = The IP address of eth1 on hasys04 is 192.168.23.104         = -->
  <!-- =============================================================== -->
  <PhysicalNetwork physicalNetworkName="db2_private_network_0"
                   physicalNetworkProtocol="ip">

    <Interface interfaceName="eth1" clusterNodeName="hasys01">
      <IPAddress baseAddress="192.168.23.101"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

    <Interface interfaceName="eth1" clusterNodeName="hasys02">
      <IPAddress baseAddress="192.168.23.102"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

    <Interface interfaceName="eth1" clusterNodeName="hasys03">
      <IPAddress baseAddress="192.168.23.103"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

    <Interface interfaceName="eth1" clusterNodeName="hasys04">
      <IPAddress baseAddress="192.168.23.104"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

  </PhysicalNetwork>

  <!-- =============================================================== -->
  <!-- = List the computers (cluster nodes) in the cluster domain.   = -->
  <!-- =============================================================== -->
  <ClusterNode clusterNodeName="hasys01"/>
  <ClusterNode clusterNodeName="hasys02"/>
  <ClusterNode clusterNodeName="hasys03"/>
  <ClusterNode clusterNodeName="hasys04"/>

</ClusterDomain>


<!-- ================================================================= -->
```

```
  <!-- = The failover policy specifies the order in which the cluster  = -->
  <!-- = domain nodes should fail over.                                = -->
  <!-- ================================================================= -->
  <FailoverPolicy>
     <Mutual />
  </FailoverPolicy>


  <!-- ================================================================= -->
  <!-- = Specify all the details of the database partitions.           = -->
  <!-- ================================================================= -->
  <DB2PartitionSet>

    <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
       <VirtualIPAddress baseAddress="19.126.124.251"
                         subnetMask="255.255.255.0"
                         networkName="db2_public_network_0"/>
       <Mount filesystemPath="/hafs/db2inst1/NODE0000"/>
       <MutualPair systemPairNode1="hasys01" systemPairNode2="hasys02" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="1" instanceName="db2inst1">
       <Mount filesystemPath="/hafs/db2inst1/NODE0001"/>
       <MutualPair systemPairNode1="hasys02" systemPairNode2="hasys01" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="2" instanceName="db2inst1">
       <Mount filesystemPath="/hafs/db2inst1/NODE0002"/>
       <MutualPair systemPairNode1="hasys03" systemPairNode2="hasys04" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="3" instanceName="db2inst1">
       <Mount filesystemPath="/hafs/db2inst1/NODE0003"/>
       <MutualPair systemPairNode1="hasys04" systemPairNode2="hasys03" />
    </DB2Partition>

  </DB2PartitionSet>


  <!-- ================================================================= -->
  <!-- = List of databases to be configured for High Availability      = -->
  <!-- ================================================================= -->
  <HADBSet instanceName="db2inst1">
    <HADB databaseName = "SAMPLE" />
    <HADB databaseName = "MYDB" />
  </HADBSet>


</DB2Cluster>
```

db2ha_sample_DPF_NPlusM.xml:

The sample file `db2ha_sample_DPF_NPlusM.xml` is an example of an XML input file that you pass to DB2 High Availability Instance Configuration Utility (db2haicu) to specify a new *cluster domain*. `db2ha_sample_DPF_NPlusM.xml` is located in the `sqllib/samples/ha/xml` directory.

Features

The `db2ha_sample_DPF_NPlusM.xml` sample domonstrates how to use db2haicu with an XML input file to define a cluster domain with the following details:
- quorum device: network
- computers in the cluster (cluster domain nodes): four

- failover policy: N Plus M
- database partitions: two
- virtual (service) IP addresses: one
- shared mount points for failover: four

XML source

```
<!-- ================================================================= -->
<!-- = Use the DB2 High Availability Instance Configuration Utility  = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify      = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP)       = -->
<!-- = Base Component as the cluster manager.                        = -->
<!-- ================================================================= -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="db2ha.xsd"
            clusterManagerName="TSA"
            version="1.0">

  <!-- ================================================================= -->
  <!-- = Create a cluster domain named db2HAdomain.                    = -->
  <!-- ================================================================= -->
  <ClusterDomain domainName="db2HAdomain">

    <!-- =============================================================== -->
    <!-- = Specify a network quorum device (IP address: 19.126.4.5).   = -->
    <!-- = The IP must be pingable at all times by each of the cluster = -->
    <!-- = domain nodes.                                               = -->
    <!-- =============================================================== -->
    <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

    <!-- =============================================================== -->
    <!-- = Create a network named db2_public_network_0 with an IP      = -->
    <!-- = network protocol.                                           = -->
    <!-- = This network contains four computers: hasys01, hasys02,     = -->
    <!-- = hasys03, and hasys04.                                       = -->
    <!-- = Each computer has a network interface card called eth0.     = -->
    <!-- = The IP address of eth0 on hasys01 is 19.126.124.30          = -->
    <!-- = The IP address of eth0 on hasys02 is 19.126.124.31          = -->
    <!-- = The IP address of eth0 on hasys03 is 19.126.124.32          = -->
    <!-- = The IP address of eth0 on hasys04 is 19.126.124.33          = -->
    <!-- =============================================================== -->
    <PhysicalNetwork physicalNetworkName="db2_public_network_0"
                     physicalNetworkProtocol="ip">

      <Interface interfaceName="eth0" clusterNodeName="hasys01">
        <IPAddress baseAddress="19.126.124.30"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys02">
        <IPAddress baseAddress="19.126.124.31"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys03">
        <IPAddress baseAddress="19.126.124.32"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
      </Interface>

      <Interface interfaceName="eth0" clusterNodeName="hasys04">
        <IPAddress baseAddress="19.126.124.33"
                   subnetMask="255.255.255.0"
                   networkName="db2_public_network_0"/>
    </Interface>

  </PhysicalNetwork>

  <!-- =============================================================== -->
  <!-- = Create a network named db2_private_network_0 with an IP     = -->
  <!-- = network protocol.                                           = -->
  <!-- = This network contains four computers: hasys01, hasys02,     = -->
  <!-- = hasys03, and hasys04 (same as db2_public_network_0.)        = -->
  <!-- = In addition to eth0, each computer has a network interface  = -->
  <!-- = card called eth1.                                           = -->
  <!-- = The IP address of eth1 on hasys01 is 192.168.23.101         = -->
  <!-- = The IP address of eth1 on hasys02 is 192.168.23.102         = -->
  <!-- = The IP address of eth1 on hasys03 is 192.168.23.103         = -->
  <!-- = The IP address of eth1 on hasys04 is 192.168.23.104         = -->
  <!-- =============================================================== -->
  <PhysicalNetwork physicalNetworkName="db2_private_network_0"
                   physicalNetworkProtocol="ip">

    <Interface interfaceName="eth1" clusterNodeName="hasys01">
      <IPAddress baseAddress="192.168.23.101"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

    <Interface interfaceName="eth1" clusterNodeName="hasys02">
      <IPAddress baseAddress="192.168.23.102"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

    <Interface interfaceName="eth1" clusterNodeName="hasys03">
      <IPAddress baseAddress="192.168.23.103"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

    <Interface interfaceName="eth1" clusterNodeName="hasys04">
      <IPAddress baseAddress="192.168.23.104"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

  </PhysicalNetwork>

  <!-- =============================================================== -->
  <!-- = List the computers (cluster nodes) in the cluster domain.   = -->
  <!-- =============================================================== -->
  <ClusterNode clusterNodeName="hasys01"/>
  <ClusterNode clusterNodeName="hasys02"/>
  <ClusterNode clusterNodeName="hasys03"/>
  <ClusterNode clusterNodeName="hasys04"/>

</ClusterDomain>


<!-- ================================================================= -->
<!-- = The failover policy specifies the order in which the cluster  = -->
<!-- = domain nodes should fail over.                                = -->
<!-- ================================================================= -->
<FailoverPolicy>
   <NPlusM />
</FailoverPolicy>


<!-- ================================================================= -->
<!-- = Specify all the details of the database partitions            = -->
```

```
  <!-- ================================================================= -->
  <DB2PartitionSet>

    <DB2Partition dbpartitionnum="0" instanceName="db2inst1">
       <VirtualIPAddress baseAddress="19.126.124.250"
                         subnetMask="255.255.255.0"
                         networkName="db2_public_network_0"/>
       <Mount filesystemPath="/ha_dpf1/db2inst1/NODE0000"/>
       <Mount filesystemPath="/hafs/NODE0000"/>
       <NPlusMNode standbyNodeName="hasys03" />
    </DB2Partition>

    <DB2Partition dbpartitionnum="1" instanceName="db2inst1">
       <Mount filesystemPath="/ha_dpf1/db2inst1/NODE0001"/>
       <Mount filesystemPath="/hafs/NODE0001"/>
       <NPlusMNode standbyNodeName="hasys04" />
    </DB2Partition>

  </DB2PartitionSet>

</DB2Cluster>
```

db2ha_sample_HADR.xml:

The sample file db2ha_sample_DPF_HADR.xml is an example of an XML input file that you pass to DB2 High Availability Instance Configuration Utility (db2haicu) to specify a new *cluster domain*. db2ha_sample_HADR.xml is located in the sqllib/samples/ha/xml directory.

Features

The db2ha_sample_HADR.xml sample domonstrates how to use db2haicu with an XML input file to define a cluster domain with the following details:
- quorum device: network
- computers in the cluster (cluster domain nodes): two
- failover policy: HADR
- database partitions: one
- virtual (service) IP addresses: none
- shared mount points for failover: none

XML source

```
<!-- ================================================================= -->
<!-- = Use the DB2 High Availability Instance Configuration Utility  = -->
<!-- = (db2haicu) XML schema definition, db2ha.xsd, and specify      = -->
<!-- = IBM Tivoli System Automation for Multiplatforms (SA MP)       = -->
<!-- = Base Component as the cluster manager.                        = -->
<!-- ================================================================= -->
<DB2Cluster xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="db2ha.xsd"
           clusterManagerName="TSA"
           version="1.0">

  <!-- ================================================================= -->
  <!-- = Create a cluster domain named db2HAdomain.                    = -->
  <!-- ================================================================= -->
  <ClusterDomain domainName="db2HAdomain">

    <!-- ================================================================= -->
    <!-- = Specify a network quorum device (IP address: 19.126.4.5).   = -->
    <!-- = The IP must be pingable at all times by each of the cluster = -->
    <!-- = domain nodes.                                               = -->
  <!-- ================================================================= -->
  <Quorum quorumDeviceProtocol="network" quorumDeviceName="19.126.4.5"/>

  <!-- ================================================================= -->
  <!-- = Create a network named db2_public_network_0 with an IP      = -->
  <!-- = network protocol.                                           = -->
  <!-- = This network contains two computers: hasys01 and hasys02.   = -->
  <!-- = Each computer has a network interface card called eth0.     = -->
  <!-- = The IP address of eth0 on hasys01 is 19.126.52.139          = -->
  <!-- = The IP address of eth0 on hasys01 is 19.126.52.140          = -->
  <!-- ================================================================= -->
  <PhysicalNetwork physicalNetworkName="db2_public_network_0"
                   physicalNetworkProtocol="ip">

    <Interface interfaceName="eth0" clusterNodeName="hasys01">
      <IPAddress baseAddress="19.126.52.139"
                 subnetMask="255.255.255.0"
                 networkName="db2_public_network_0"/>
    </Interface>

    <Interface interfaceName="eth0" clusterNodeName="hasys02">
      <IPAddress baseAddress="19.126.52.140"
                 subnetMask="255.255.255.0"
                 networkName="db2_public_network_0"/>
    </Interface>

  </PhysicalNetwork>

  <!-- ================================================================= -->
  <!-- = Create a network named db2_private_network_0 with an IP     = -->
  <!-- = network protocol.                                           = -->
  <!-- = This network contains two computers: hasys01 and hasys02.   = -->
  <!-- = In addition to eth0, each computer has a network interface  = -->
  <!-- = card called eth1.                                           = -->
  <!-- = The IP address of eth1 on hasys01 is 192.168.23.101         = -->
  <!-- = The IP address of eth1 on hasys02 is 192.168.23.102         = -->
  <!-- ================================================================= -->
  <PhysicalNetwork physicalNetworkName="db2_private_network_0"
                   physicalNetworkProtocol="ip">

    <Interface interfaceName="eth1" clusterNodeName="hasys01">
      <IPAddress baseAddress="192.168.23.101"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

    <Interface interfaceName="eth1" clusterNodeName="hasys02">
      <IPAddress baseAddress="192.168.23.102"
                 subnetMask="255.255.255.0"
                 networkName="db2_private_network_0"/>
    </Interface>

  </PhysicalNetwork>

  <!-- ================================================================= -->
  <!-- = List the computers (cluster nodes) in the cluster domain.   = -->
  <!-- ================================================================= -->
  <ClusterNode clusterNodeName="hasys01"/>
  <ClusterNode clusterNodeName="hasys02"/>

</ClusterDomain>


<!-- ===================================================================== -->
<!-- = The failover policy specifies the order in which the cluster    = -->
<!-- = domain nodes should fail over.                                  = -->
<!-- ===================================================================== -->
```

```
   <FailoverPolicy>
      <HADRFailover />
   </FailoverPolicy>


   <!-- ================================================================= -->
   <!-- = Specify all the details of the database partitions           = -->
   <!-- ================================================================= -->
   <DB2PartitionSet>
     <DB2Partition dbpartitionnum="0" instanceName="db2inst1" />
   </DB2PartitionSet>


   <!-- ================================================================= -->
   <!-- = List of HADR databases                                        = -->
   <!-- ================================================================= -->
   <HADRDBSet>
     <HADRDB databaseName="HADRDB"
             localInstance="db2inst1"
             remoteInstance="db2inst1"
             localHost="hasys01"
             remoteHost="hasys02" />
   </HADRDBSet>

</DB2Cluster>
```

DB2 High Availability Instance Configuration Utility (db2haicu) prerequisites

There is a set of tasks you must perform before using DB2 High Availability Instance Configuration Utility (db2haicu).

General

Before a database manager instance owner can run db2haicu, a user with root authority must run the preprpnode command.

preprpnode is part of the Reliable Scalable Cluster Technology (RSCT) fileset for AIX and the RSCT package for Linux. preprpnode handles initializing the nodes for intracluster communication. The preprpnode command is run as a part of setting up the cluster. For more information about preprpnode, see:

- `preprpnode` Command (AIX)
- RSCT for Linux Technical Reference - preprpnode

For more information about RSCT, see RSCT Administration Guide - What is RSCT?

Before running db2haicu, a database manager instance owner must perform the following tasks:

- Synchronize services files on all machines that will be added to the cluster.
- Run the db2profile script for the database manager instance that will be used to create the cluster domain.
- Start the database manager using the db2start command.

DB2 High Availability Disaster Recovery (HADR)

If you will be using HADR functionality, perform the following tasks:

- Ensure all DB2 High Availability Disaster Recovery (HADR) databases are started in their respective primary and standby database roles, and that all HADR primary-standby database pairs are in peer state.

- Configure **hadr_peer_window** for all HADR databases to a value of at least 120 seconds.
- Disable DB2 fault monitor.

Partitioned database environment

If you have multiple database partitions to configure for high availability, perform the following steps:

- Configure the DB2_NUM_FAILOVER_NODES registry variable on all machines that will be added to the cluster domain.
- (Optional) Activate the database before running db2haicu.

Creating a cluster domain using DB2 High Availability Instance Configuration Utility (db2haicu)

When you run DB2 High Availability Instance Configuration Utility (db2haicu) for the first time for a database manager instance, db2haicu creates a model of your cluster, called a *cluster domain*.

Database paths detected automatically by DB2 High Availability Instance Configuration Utility (db2haicu):

When you run DB2 High Availability Instance Configuration Utility (db2haicu) for the first time, db2haicu will search your database system for database configuration information that is related to cluster configuration.

Single database partition environment

In a single database partition environment, db2haicu automatically detects the paths:

- Instance home directory path
- Audit log path
- Audit archive log path
- Sync point manager (SPM) log path
- DB2 diagnostic log (db2diag log file) path
- Database related paths:
  - Database log path
  - Database table space container path
  - Database table space directory path
  - Local database directory

Multiple database partition environment

In a multiple database partition environment, db2haicu automatically detects only the following paths:

- Database log path
- Database table space container path
- Database table space directory path
- Local database directory

Maintaining a cluster domain using DB2 High Availability Instance Configuration Utility (db2haicu)

When you are modifying the cluster domain model of your clustered environment using db2haicu, the database manager propagates the related changes to your database manager instance and cluster configuration.

Before you begin

Before you can configure your clustered environment using db2haicu, you must create and configure a cluster domain. For more information, see "Creating a cluster domain using DB2 High Availability Instance Configuration Utility (db2haicu)" on page 110

About this task

db2haicu maintenance tasks include adding cluster elements such as databases or cluster nodes to the cluster domain, and removing elements from the cluster domain. db2haicu maintenance tasks also include modifying the details of cluster domain elements such as the failover policy for the database manager instance.

Procedure

1. Run db2haicu

   When you run db2haicu in maintenance mode, db2haicu presents you with a list of operations you can perform on the cluster domain:
   - Add or remove cluster nodes (machine identified by hostname)
   - Add or remove a network interface (network interface card)
   - Add or remove database partitions (partitioned database environment only)
   - Add or remove a DB2 High Availability Disaster Recovery (HADR) database
   - Add or remove a highly available database
   - Add or remove a mount point
   - Add or remove an IP address
   - Add or remove a non-critical path
   - Move database partitions and HADR databases for scheduled maintenance
   - Change failover policy for the current instance
   - Create a new quorum device for the cluster domain
   - Destroy the cluster domain
2. Select a task to perform, and answer subsequent questions that db2haicu presents.

Results

The database manager uses the information in the cluster domain to coordinate with your cluster manager. When you configure your database and cluster elements using db2haicu then those elements are included in integrated and automated cluster configuration and administration provided by the DB2 High Availability (HA) Feature. When you use db2haicu to make a database manager instance configuration change, the database manager makes the required cluster manager configuration change for you so that you do not have to make a subsequent call to your cluster manager.

What to do next

DB2 High Availability Instance Configuration Utility (db2haicu) does not have a separate diagnostic log. You can investigate and diagnose db2haicu errors using the database manager diagnostic log, db2diag log file, and the db2pd tool. For more information, see: "Troubleshooting DB2 High Availability Instance Configuration Utility (db2haicu)"

Troubleshooting DB2 High Availability Instance Configuration Utility (db2haicu)

DB2 High Availability Instance Configuration Utility (db2haicu) does not have a separate diagnostic log. You can investigate and diagnose db2haicu errors using the database manager diagnostic log, db2diag log file, and the db2pd tool.

DB2 High Availability Instance Configuration Utility (db2haicu) restrictions

There are some restrictions for using DB2 High Availability Instance Configuration Utility (db2haicu).

- "Software and hardware"
- "Configuration tasks"
- "Usage notes"
- "Recommendations" on page 113

Software and hardware

- Currently, IBM Tivoli System Automation for Multiplatforms (SA MP) Version 2.2, fixpack 3 is the only cluster manager that db2haicu supports.
- Reliable Scalable Cluster Technology (RSCT) Version 2.4.7.3 is also required. For more information about RSCT, see: RSCT Administration Guide - What is RSCT?
- db2haicu does not support IP version 6.

Configuration tasks

You cannot perform the following tasks using db2haicu:

- You cannot configure automatic client reroute using db2haicu.
- When you upgrade from DB2 Database for Linux, UNIX, and Windows Version 9 to IBM Data Server Version 9.5, or from Version 9.5 to a later version, you cannot use db2haicu to to migrate your cluster configuration. To migrate a cluster configuration, you must perform the following steps:
  1. Delete the existing cluster domain (if one exists)
  2. Upgrade the database server
  3. Create a new cluster domain using db2haicu

Usage notes

Consider the following db2haicu usage notes when planning your cluster configuration and administration activities:

- Even though db2haicu performs some administration tasks that normally require root authority, db2haicu runs with the privileges of the database manager instance owner. db2haicu initialization, performed by a root user, enables db2haicu to carry out the required configuration changes despite having only instance owner privileges.

- When you create a new cluster domain, db2haicu does not verify that the name you specify for the new cluster domain is valid. For example, db2haicu does not confirm that the name is a valid length, or contains valid characters, or that is not the same name as an existing cluster domain.
- db2haicu does not verify or validate information that a user specifies and that is passed to a cluster manager. Because db2haicu cannot be aware of all cluster manager restrictions with respect to cluster object names, for example, db2haicu passes text to the cluster manager without validating it for things like valid characters, or length.
- If an error happens and db2haicu fails while you are creating and configuring a new cluster domain, you must perform the following steps:
  1. Remove the resource groups of the partially created cluster domain by running db2haicu using the **-delete** parameter
  2. Recreate the new cluster domain by calling db2haicu again.
- When you run db2haicu with the **-delete** parameter, db2haicu deletes the resource groups associated with the current database manager instance immediately, without confirming whether thoseresource groups are locked.
- To remove resource groups associated with the database manager instances of a DB2 High Availability Disaster Recovery (HADR) primary database, standby database pair, perform the following steps:
  1. Run db2haicu with the **-delete** parameter against the database manager instance of the HADR standby database first.
  2. Also run db2haicu with the **-delete** parameter against the database manager instance of the HADR primary database.
- If a cluster operation you attempt to perform using db2haicu times out, db2haicu will not return an error to you. When a cluster operation times out, you will not know that the operation timed out unless you review diagnostic logs after making the db2haicu call; or unless a subsequent cluster action fails, and while investigating that subsequent failure, you determine that the original cluster operation timed out.
- If you attempt to change the failover policy for a given database instance to active-passive, there is one condition under which that configuration operation will fail, but for which db2haicu will not return a error to you. If you specify a machine that is currently offline to be the *active* machine, db2haicu will not make that machine the active machine, but db2haicu will not return an error indicating that the change did not succeed.

Recommendations

The following is a list of recommendations for configuration your cluster, and your database manager instances when using db2haicu.

- When you add new mount points for the cluster by adding entries to /etc/fstab, use the **noauto** option to prevent the mount points from being automatically mounted on more than one machine in the cluster. For example:

```
dev/vpatha1      /db/svtpdb/NODE0010        ext3  noauto 0 0
```

DB2 cluster manager API

The DB2 cluster manager API defines a set of functions that enable the database manager to communicate configuration changes to the cluster manager.

Supported cluster management software

Cluster managing software enables the transfer of DB2 database operations from a failed primary database on one node of the cluster to a secondary database on another node in the cluster.

DB2 database supports the following cluster managing software:

- High Availability Cluster Multi-Processing (HACMP), for AIX

  For detailed information about HACMP/ES, see the white paper entitled "IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES", which is available from the IBM Software Library web site (http://www.ibm.com/software/sw-library/).
- Tivoli System Automation for Linux.

  For detailed information about Tivoli System Automation, see the white paper entitled "Highly Available DB2 Universal Database using Tivoli System Automation for Linux", which is available from the IBM Software Library web site (http://www.ibm.com/software/sw-library/).
- Microsoft® Cluster Server, for Windows operating systems

  For information about Microsoft Cluster Server see the following white paper which is available from the IBM Software Library web site (http://www.ibm.com/software/sw-library/): "Implementing IBM DB2 Universal Database V8.1 Enterprise Server Edition with Microsoft Cluster Server".
- Sun Cluster, or VERITAS Cluster Server, for the Solaris operating system.

  For information about Sun Cluster, see the white paper entitled " DB2 Universal Database and High Availability on Sun Cluster 3.X", which is available from the IBM Software Library web site (http://www.ibm.com/software/sw-library/). For information about VERITAS Cluster Server, see the white paper entitled "DB2 UDB and High Availability with VERITAS Cluster Server", which is available from the "IBM Support and downloads" Web site (http://www.ibm.com/support/docview.wss?uid=swg21045033).
- Multi-Computer/ServiceGuard, for Hewlett-Packard

High Availability Cluster Multi-Processing for AIX

High Availability Cluster Multi-Processing (HACMP) for AIX is cluster managing software. The nodes in HACMP clusters exchange messages called heartbeats, or keepalive packets. If a node stops sending these messages, HACMP invokes failover across the other nodes in the cluster; and when the node that failed is repaired, HACMP reintegrates it back into the cluster.

There are two types of events: standard events that are anticipated within the operations of HACMP, and user-defined events that are associated with the monitoring of parameters in hardware and software components.

One of the standard events is the node_down event. This is when a node in the cluster has failed, and HACMP has initiated failover across the other nodes in the cluster. When planning what should be done as part of the recovery process, HACMP allows two failover options: hot (or idle) standby, and mutual takeover.

Note: When using HACMP, ensure that DB2 instances are not started at boot time by using the db2iauto utility as follows:

```
db2iauto -off InstName
```

where *InstName* is the login name of the instance.

Cluster Configuration

In a *hot standby* configuration, the AIX processor node that is the takeover node *is not* running any other workload. In a *mutual takeover* configuration, the AIX processor node that is the takeover node *is* running other workloads.

Generally, in a partitioned database environment, DB2 database runs in mutual takeover mode with database partitions on each node. One exception is a scenario in which the catalog partition is part of a hot standby configuration.

One planning consideration is how to manage big clusters. It is easier to manage a small cluster than a big one; however, it is also easier to manage one big cluster than many smaller ones. When planning, consider how your applications will be used in your cluster environment. If there is a single, large, homogeneous application running, for example, on 16 nodes, it is probably easier to manage the configuration as a single cluster rather than as eight two-node clusters. If the same 16 nodes contain many different applications with different networks, disks, and node relationships, it is probably better to group the nodes into smaller clusters. Keep in mind that nodes integrate into an HACMP cluster one at a time; it will be faster to start a configuration of multiple clusters rather than one large cluster. HACMP supports both single and multiple clusters, as long as a node and its backup are in the same cluster.

HACMP failover recovery allows predefined (also known as *cascading*) assignment of a resource group to a physical node. The failover recovery procedure also allows floating (or *rotating*) assignment of a resource group to a physical node. IP addresses, and external disk volume groups, or file systems, or NFS file systems, and application servers within each resource group specify either an application or an application component, which can be manipulated by HACMP between physical nodes by failover and reintegration. Failover and reintegration behavior is specified by the type of resource group created, and by the number of nodes placed in the resource group.

For example, consider a DB2 database partition (logical node). If its log and table space containers were placed on external disks, and other nodes were linked to those disks, it would be possible for those other nodes to access these disks and to restart the database partition (on a takeover node). It is this type of operation that is automated by HACMP. HACMP can also be used to recover NFS file systems used by DB2 instance main user directories.

Read the HACMP documentation thoroughly as part of your planning for recovery with DB2 database in a partitioned database environment. You should read the Concepts, Planning, Installation, and Administration guides, then build the recovery architecture for your environment. For each subsystem that you have identified for recovery, based on known points of failure, identify the HACMP clusters that you need, as well as the recovery nodes (either hot standby or mutual takeover).

It is strongly recommended that both disks and adapters be mirrored in your external disk configuration. For DB2 physical nodes that are configured for HACMP, care is required to ensure that nodes on the volume group can vary from the shared external disks. In a mutual takeover configuration, this arrangement requires some additional planning, so that the paired nodes can access each other's volume groups without conflicts. In a partitioned database environment, this means that all container names must be unique across all databases.

One way to achieve uniqueness is to include the database partition number as part of the name. You can specify a node expression for container string syntax when creating either SMS or DMS containers. When you specify the expression, the node number can be part of the container name or, if you specify additional arguments, the results of those arguments can be part of the container name. Use the argument ″ $N″ (blank]$N) to indicate the node expression. The argument must occur at the end of the container string, and can only be used in one of the following forms:

Table 7. Arguments for Creating Containers. The node number is assumed to be five.

Syntax	Example	Value
blank]$N	″ $N″	5
blank]$N+ number]	″ $N+1011″	1016
blank]$N% number]	″ $N%3″	2
blank]$N+ number]% number]	″ $N+12%13″	4
blank]$N% number]+ number]	″ $N%3+20″	22
Note: 1. % is modulus. 2. In all cases, the operators are evaluated from left to right.		

Following are some examples of how to create containers using this special argument:

- Creating containers for use on a two-node system.

  ```
  CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
     (device '/dev/rcont $N' 20000)
  ```

  The following containers would be used:

  ```
  /dev/rcont0   - on Node 0
  /dev/rcont1   - on Node 1
  ```

- Creating containers for use on a four-node system.

  ```
  CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
     (file '/DB2/containers/TS2/container $N+100' 10000)
  ```

  The following containers would be used:

  ```
  /DB2/containers/TS2/container100   - on Node 0
  /DB2/containers/TS2/container101   - on Node 1
  /DB2/containers/TS2/container102   - on Node 2
  /DB2/containers/TS2/container103   - on Node 3
  ```

- Creating containers for use on a two-node system.

  ```
  CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
     ('/TS3/cont $N%2, '/TS3/cont $N%2+2')
  ```

  The following containers would be used:

  ```
  /TS3/cont0   - on Node 0
  /TS3/cont2   - on Node 0
  /TS3/cont1   - on Node 1
  /TS3/cont3   - on Node 1
  ```

Configuring DB2 Database Partitions for HACMP

Once configured, each database partition in an instance is started by HACMP, one physical node at a time. Multiple clusters are recommended for starting parallel

DB2 configurations that are larger than four nodes. Note that in a 64-node parallel DB2 configuration, it is faster to start 32 two-node HACMP clusters in parallel, than four 16-node clusters.

A script file is packaged with DB2 Enterprise Server Edition to assist in configuring for HACMP failover or recovery in either hot standby or mutual takeover nodes. The script file is called rc.db2pe.ee for a single node and rc.db2pe.eee for multiple nodes. They are located in the sqllib/samples/hacmp/es directory. Copy the appropriate file to /usr/bin on each system in the HACMP cluster and rename it to rc.db2pe.

In addition, DB2 buffer pool sizes can be customized during failover in mutual takeover configurations from within rc.db2pe. (Buffer pool sizes can be configured to ensure proper resource allocation when two database partitions run on one physical node.)

HACMP Event Monitoring and User-defined Events

Initiating a failover operation if a process dies on a given node, is an example of a user-defined event. Events must be configured manually as a user defined event as part of the cluster setup.

For detailed information on the implementation and design of highly available IBM DB2 database environments see the IBM Software Library web site (http://www.ibm.com/software/sw-library/).

IBM Tivoli System Automation for Multiplatforms (Linux and AIX)

IBM Tivoli System Automation for Multiplatforms (Tivoli SAM) is cluster managing software that facilitates automatic switching of users, applications, and data from one database system to another in a cluster. Tivoli SAM automates control of IT resources such as processes, file systems, and IP addresses.

Tivoli SAM provides a framework to automatically manage the availability of what are known as resources. Examples of resources are:

- Any piece of software for which start, monitor, and stop scripts can be written to control
- Any network interface card (NIC) to which Tivoli SAM has been granted access. That is, Tivoli SAM will manage the availability of any IP address that a user wants to use by floating that IP address amongst NICs that it has been granted access to.

For example, both a DB2 instance and the High Availability Disaster Recovery feature, have start, stop, and monitor commands. Therefore, Tivoli SAM scripts can be written to automatically manage these resources. Resources that are closely related (for example, ones that collectively run on the same node at the same time) are called a *resource group*.

DB2 resources

In a single-partition DB2 environment, a single DB2 instance is running on a server. This DB2 instance has local access to data (its own executable image as well as databases owned by the instance). If this DB2 instance is made accessible to remote clients, an unused IP address must be assigned to this DB2 instance.

The DB2 instance, the local data, and the IP address are all considered resources, which must be automated by Tivoli SAM. Since these resources are closely related (for example, they collectively run on the same node at the same time), they are called a resource group.

The entire resource group is collocated on one node in the cluster. In the case of a failover, the entire resource group is started on another node.

The following dependencies exist between the resources in the group:
- The DB2 instance must be started after the local disk
- The DB2 instance must be stopped before the local disk
- The HA IP address must be collocated with the instance

Disk storage

The DB2 database can utilize these resources for local data storage:
- Raw disk (for example, /dev/sda1)
- Logical volume managed by Logical Volume Manager (LVM)
- File system (for example, ext3, jfs)

DB2 data can be stored either entirely on one or more raw disks, entirely on logical volumes, entirely on file systems, or on a mixture of all three. Any executables need to reside on a file system of some sort.

DB2 database requirements for the HA IP address

The DB2 database has no special requirements for the IP address. It is not necessary to define a highly available IP address in order for the instance to be considered highly available. However, it is important to remember that the IP address that is protected (if any) is the client's access point to the data, and as such, this address must be well known by all clients. In practice, it is recommended that this IP address be the one that is used by the clients in their CATALOG TCPIP NODE commands.

Tivoli SAM resource groups

IBM Tivoli System Automation for Multiplatforms is a product that provides high availability by automating resources such as processes, applications, IP addresses, and others in Linux-based clusters. To automate an IT resource (such as an IP address), the resource must be defined to Tivoli SAM. Furthermore, these resources must all be contained in at least one resource group. If these resources are always required to be hosted on the same machine, they should all be placed in the same resource group.

Every application needs to be defined as a resource in order to be managed and automated with Tivoli SAM. Application resources are usually defined in the generic resource class IBM.Application. In this resource class, there are several attributes that define a resource, but at least three of them are application-specific:
- StartCommand
- StopCommand
- MonitorCommand

These commands may be scripts or binary executables.

Setting up Tivoli SAM with your DB2 environment

For detailed configuration information to help you set up Tivoli SAM to work with your DB2 environment, search for "Tivoli System Automation" on the IBM Software Library web site (http://www.ibm.com/software/sw-library/).

Microsoft Failover Clustering support (Windows)

Microsoft Failover Clustering supports clusters of servers on Windows operating systems. It automatically detects and responds to server or application failure, and can balance server workloads.

Introduction

Microsoft Failover Clustering is a feature of Windows Server operating systems. It is the software that supports the connection of two servers (up to four servers in DataCenter Server) into a cluster for high availability and easier management of data and applications. Failover Clustering can also automatically detect and recover from server or application failures. It can be used to move server workloads to balance machine utilization and to provide for planned maintenance without downtime.

The following DB2 products have support for Failover Clustering:
- DB2 Workgroup Server Edition
- DB2 Enterprise Server Edition (DB2 ESE)
- DB2 Connect Enterprise Edition (DB2 CEE)

DB2 Failover Clustering Components

A cluster is a configuration of two or more nodes, each of which is an independent computer system. The cluster appears to network clients as a single server.

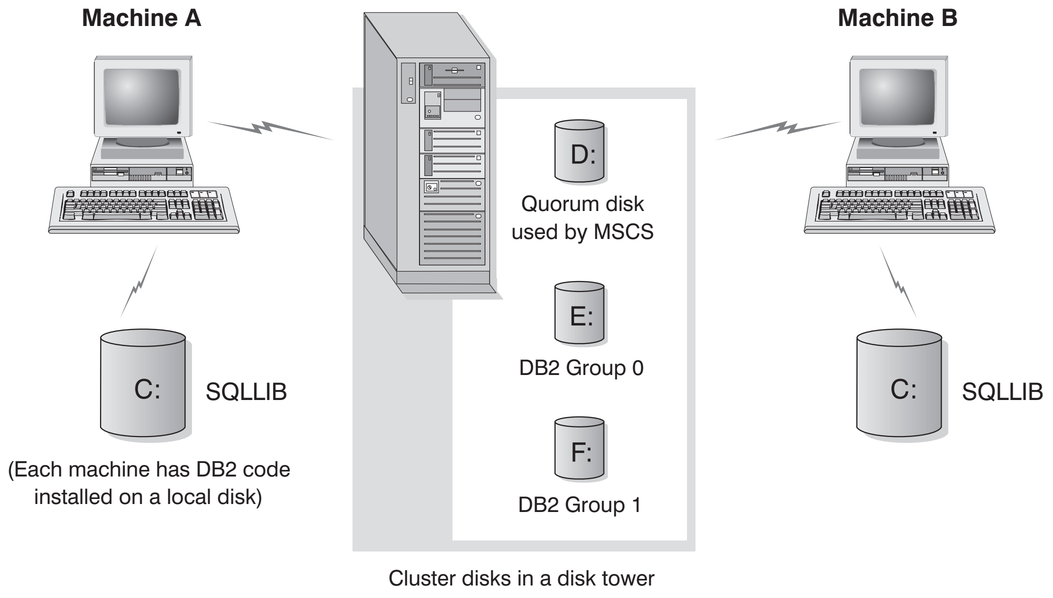


Figure 3. Example Failover Clustering Configuration

The nodes in an Failover Clustering cluster are connected using one or more shared storage buses and one or more physically independent networks. The network that connects only the servers but does not connect the clients to the cluster is referred to as a *private* network. The network that supports client connections is referred to as the *public* network. There are one or more local disks on each node. Each shared storage bus attaches to one or more disks. Each disk on the shared bus is owned by only one node of the cluster at a time. The DB2 software resides on the local disk. DB2 database files (for example tables, indexes, log files) reside on the shared disks. Because Failover Clustering does not support the use of raw partitions in a cluster, it is not possible to configure DB2 to use raw devices in an Failover Clustering environment.

The DB2 Resource

In a Failover Clustering environment, a resource is an entity that is managed by the clustering software. For example, a disk, an IP address, or a generic service can be managed as a resource. DB2 integrates with Failover Clustering by creating its own resource type called DB2 Server. Each DB2 Server resource manages a DB2 instance, and when running in a partitioned database environment, each DB2 Server resource manages a database partition. The name of the DB2 Server resource is the instance name, although in the case of a partitioned database environment, the name of the DB2 Server resource consists of both the instance name and the database partition (or node) number.

Pre-online and Post-online Script

You can run scripts both before and after a DB2 resource is brought online. These scripts are referred to as pre-online and post-online scripts respectively. Pre-online and post-online scripts are .BAT files that can run DB2 and system commands.

In a situation when multiple instances of DB2 might be running on the same machine, you can use the pre-online and post-online scripts to adjust the configuration so that both instances can be started successfully. In the event of a failover, you can use the post-online script to perform manual database recovery. Post-online script can also be used to start any applications or services that depend on DB2.

The DB2 Group

Related or dependent resources are organized into resource groups. All resources in a group move between cluster nodes as a unit. For example, in a typical DB2 single partition cluster environment, there will be a DB2 group that contains the following resources:

1. DB2 resource. The DB2 resource manages the DB2 instance (or node).
2. IP Address resource. The IP Address resource allows client applications to connect to the DB2 server.
3. Network Name resource. The Network Name resource allows client applications to connect to the DB2 server by using a name rather than using an IP address. The Network Name resource has a dependency on the IP Address resource. The Network Name resource is optional. (Configuring a Network Name resource can affect the failover performance.)
4. One or more Physical Disk resources. Each Physical Disk resource manages a shared disk in the cluster.

Note: The DB2 resource is configured to depend on all other resources in the same group so the DB2 server can only be started after all other resources are online.

Failover Configurations

Two types of configuration are available:
- Hot standby
- Mutual takeover

In a partitioned database environment, the clusters do not all have to have the same type of configuration. You can have some clusters that are set up to use hot standby, and others that are set up for mutual takeover. For example, if your DB2 instance consists of five workstations, you can have two machines set up to use a mutual takeover configuration, two to use a hot standby configuration, and one machine not configured for failover support.

Hot Standby Configuration

In a hot standby configuration, one machine in the Failover Clustering provides dedicated failover support, and the other machine participates in the database system. If the machine participating in the database system fails, the database server on it will be started on the failover machine. If, in a partitioned database environment, you are running multiple logical nodes on a machine and it fails, the logical nodes will be started on the failover machine. Figure 4 shows an example of a hot standby configuration.

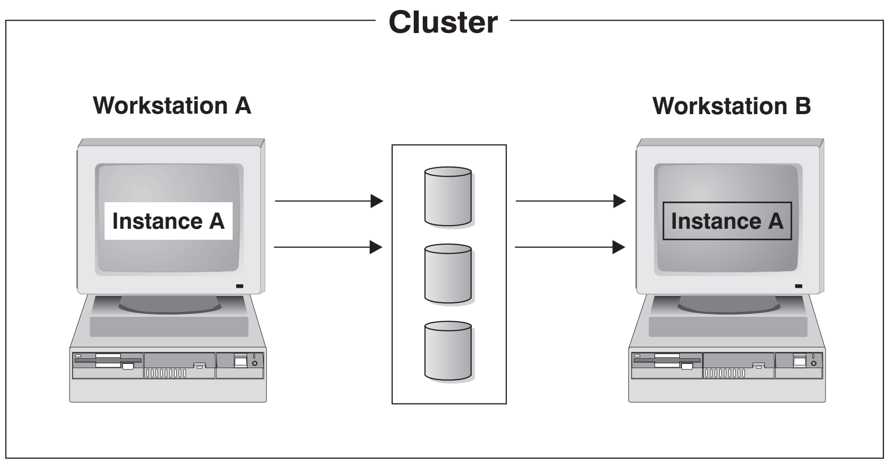


Figure 4. Hot Standby Configuration

Mutual Takeover Configuration

In a mutual takeover configuration, both workstations participate in the database system (that is, each machine has at least one database server running on it). If one of the workstations in the Failover Clustering fails, the database server on the failing machine will be started to run on the other machine. In a mutual takeover configuration, a database server on one machine can fail independently of the database server on another machine. Any database server can be active on any machine at any given point in time. Figure 5 on page 122 shows an example of a mutual takeover configuration.

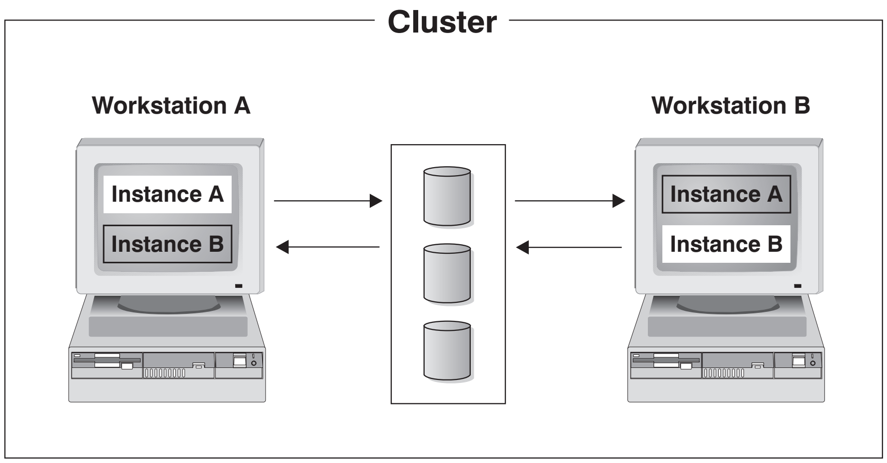


Figure 5. Mutual Takeover Configuration

For detailed information on the implementation and design of highly available IBM DB2 database environments on Windows operating systems, see the IBM Software Library web site (http://www.ibm.com/software/sw-library/).

Solaris Operating System cluster support

DB2 supports two cluster managers available for the Solaris Operating System: Sun Cluster; and Veritas Cluster Server (VCS).

For information about Sun Cluster, see the white paper entitled “ DB2 Universal Database and High Availability on Sun Cluster 3.X”, which is available from the IBM Software Library web site (http://www.ibm.com/software/sw-library/).

Note: When using Sun Cluster 3.0 or Veritas Cluster Server, ensure that DB2 instances are not started at boot time by using the db2iauto utility as follows:

```
db2iauto -off InstName
```

where *InstName* is the login name of the instance.

High Availability

The computer systems that host data services contain many distinct components, and each component has a ″mean time before failure″ (MTBF) associated with it. The MTBF is the average time that a component will remain usable. The MTBF for a quality hard drive is in the order of one million hours (approximately 114 years). While this seems like a long time, one out of 200 disks is likely to fail within a 6-month period.

Although there are a number of methods to increase availability for a data service, the most common is an HA cluster. A cluster, when used for high availability, consists of two or more machines, a set of private network interfaces, one or more public network interfaces, and some shared disks. This special configuration allows a data service to be moved from one machine to another. By moving the data service to another machine in the cluster, it should be able to continue providing access to its data. Moving a data service from one machine to another is called a *failover*, as illustrated in Figure 6 on page 123.

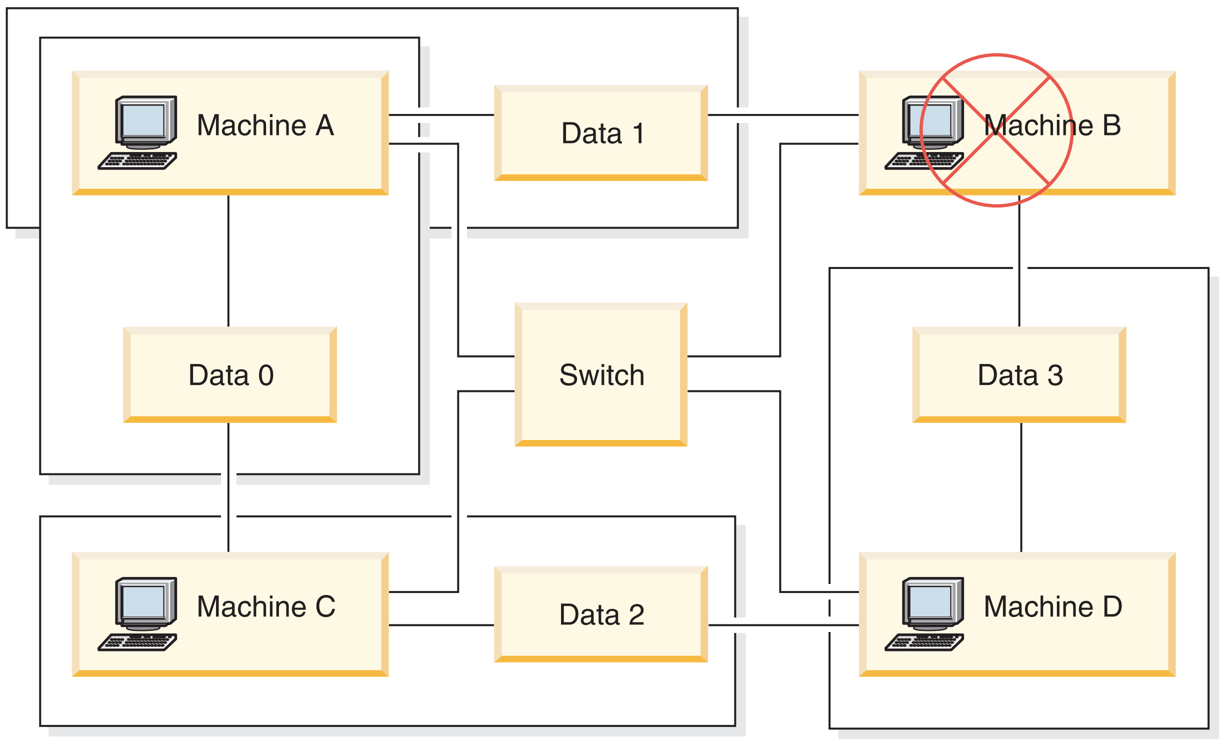


Figure 6. Failover. When Machine B fails its data service is moved to another machine in the cluster so that the data can still be accessed.

The private network interfaces are used to send *heartbeat* messages, as well as control messages, among the machines in the cluster. The public network interfaces are used to communicate directly with clients of the HA cluster. The disks in an HA cluster are connected to two or more machines in the cluster, so that if one machine fails, another machine has access to them.

A data service running on an HA cluster has one or more logical public network interfaces and a set of disks associated with it. The clients of an HA data service connect via TCP/IP to the logical network interfaces of the data service only. If a failover occurs, the data service, along with its logical network interfaces and set of disks, are moved to another machine.

One of the benefits of an HA cluster is that a data service can recover without the aid of support staff, and it can do so at any time. Another benefit is redundancy. All of the parts in the cluster should be redundant, including the machines themselves. The cluster should be able to survive any single point of failure.

Even though highly available data services can be very different in nature, they have some common requirements. Clients of a highly available data service expect the network address and host name of the data service to remain the same, and expect to be able to make requests in the same way, regardless of which machine the data service is on.

Consider a web browser that is accessing a highly available web server. The request is issued with a URL (Uniform Resource Locator), which contains both a host name, and the path to a file on the web server. The browser expects both the host name and the path to remain the same after failover of the web server. If the browser is downloading a file from the web server, and the server is failed over, the browser will need to reissue the request.

Availability of a data service is measured by the amount of time the data service is available to its users. The most common unit of measurement for availability is the percentage of ″up time″; this is often referred to as the number of ″nines″:

```
99.99% => service is down for (at most) 52.6 minutes / yr
99.999% => service is down for (at most) 5.26 minutes / yr
99.9999% => service is down for (at most) 31.5 seconds / yr
```

When designing and testing an HA cluster:

1. Ensure that the administrator of the cluster is familiar with the system and what should happen when a failover occurs.
2. Ensure that each part of the cluster is truly redundant and can be replaced quickly if it fails.
3. Force a test system to fail in a controlled environment, and make sure that it fails over correctly each time.
4. Keep track of the reasons for each failover. Although this should not happen often, it is important to address any issues that make the cluster unstable. For example, if one piece of the cluster caused a failover five times in one month, find out why and fix it.
5. Ensure that the support staff for the cluster is notified when a failover occurs.
6. Do not overload the cluster. Ensure that the remaining systems can still handle the workload at an acceptable level after a failover.
7. Check failure-prone components (such as disks) often, so that they can be replaced before problems occur.

Fault Tolerance

Another way to increase the availability of a data service is fault tolerance. A *fault tolerant* machine has all of its redundancy built in, and should be able to withstand a single failure of any part, including CPU and memory. Fault tolerant machines are most often used in niche markets, and are usually expensive to implement. An HA cluster with machines in different geographical locations has the added advantage of being able to recover from a disaster affecting only a subset of those locations.

An HA cluster is the most common solution to increase availability because it is scalable, easy to use, and relatively inexpensive to implement.

Sun Cluster 3.0 (and higher) support:

If you plan to run your DB2 database solution on a Solaris Operating System cluster, you can use Sun Cluster 3.0 to manager the cluster. A high availability agent acts as a mediator between DB2 database and Sun Cluster 3.0.

The statements made in this topic about the support for Sun Cluster 3.0 also apply to versions of Sun Cluster higher than 3.0.

DB2
HA Agent
SC3.0

Figure 7. DB2 database, Sun Cluster 3.0, and High Availability. The relationship between DB2 database, Sun Cluster 3.0 and the high availability agent.

Failover

Sun Cluster 3.0 provides high availability by enabling application failover. Each node is periodically monitored and the cluster software automatically relocates a cluster-aware application from a failed primary node to a designated secondary node. When a failover occurs, clients might experience a brief interruption in service and might have to reconnect to the server. However, they will not be aware of the physical server from which they are accessing the application and the data. By allowing other nodes in a cluster to automatically host workloads when the primary node fails, Sun Cluster 3.0 can significantly reduce downtime and increase productivity.

Multihost Disks

Sun Cluster 3.0 requires multihost disk storage. This means that disks can be connected to more than one node at a time. In the Sun Cluster 3.0 environment, multihost storage allows disk devices to become highly available. Disk devices that reside on multihost storage can tolerate single node failures since there is still a physical path to the data through the alternate server node. Multihost disks can be accessed globally through a primary node. If client requests are accessing the data through one node and that node fails, the requests are switched over to another node that has a direct connection to the same disks. A volume manager provides for mirrored or RAID 5 configurations for data redundancy of the multihost disks. Currently, Sun Cluster 3.0 supports Solstice DiskSuite and VERITAS Volume Manager as volume managers. Combining multihost disks with disk mirroring and striping protects against both node failure and individual disk failure.

Global Devices

Global devices are used to provide cluster-wide, highly available access to any device in a cluster, from any node, regardless of the device's physical location. All disks are included in the global namespace with an assigned device ID (DID) and are configured as global devices. Therefore, the disks themselves are visible from all cluster nodes.

File systems/Global File Systems

A cluster or global file system is a proxy between the kernel (on one node) and the underlying file system volume manager (on a node that has a physical connection to one or more disks). Cluster file systems are dependent on global devices with physical connections to one or more nodes. They are independent of the underlying file system and volume manager. Currently, cluster file systems can be built on UFS using either Solstice DiskSuite or VERITAS Volume Manager. The data only becomes available to all nodes if the file systems on the disks are mounted globally as a cluster file system.

Device Group

All multihost disks must be controlled by the Sun Cluster framework. Disk groups, managed by either Solstice DiskSuite or VERITAS Volume Manager, are first created on the multihost disk. Then, they are registered as Sun Cluster disk device groups. A disk device group is a type of global device. Multihost device groups are highly available. Disks are accessible through an alternate path if the node currently mastering the device group fails. The failure of the node mastering the device group does not affect access to the device group except for the time required to perform the recovery and consistency checks. During this time, all requests are blocked (transparently to the application) until the system makes the device group available.

Resource Group Manager (RGM)

The RGM, provides the mechanism for high availability and runs as a daemon on each cluster node. It automatically starts and stops resources on selected nodes according to pre-configured policies. The RGM allows a resource to be highly available in the event of a node failure or to reboot by stopping the resource on the affected node and starting it on another. The RGM also automatically starts and stops resource-specific monitors that can detect resource failures and relocate failing resources onto another node.

Data Services

The term data service is used to describe a third-party application that has been configured to run on a cluster rather than on a single server. A data service includes the application software and Sun Cluster 3.0 software that starts, stops and monitors the application. Sun Cluster 3.0 supplies data service methods that are used to control and monitor the application within the cluster. These methods run under the control of the Resource Group Manager (RGM), which uses them to start, stop, and monitor the application on the cluster nodes. These methods, along with the cluster framework software and multihost disks, enable applications to become highly available data services. As highly available data services, they can prevent significant application interruptions after any single failure within the cluster, regardless of whether the failure is on a node, on an interface component or in the application itself. The RGM also manages resources in the cluster, including network resources (logical host names and shared addresses) and application instances.

Resource Type, Resource and Resource Group

A resource type is made up of the following:
1. A software application to be run on the cluster.
2. Control programs used as callback methods by the RGM to manage the application as a cluster resource.
3. A set of properties that form part of the static configuration of a cluster.

The RGM uses resource type properties to manage resources of a particular type.

A resource inherits the properties and values of its resource type. It is an instance of the underlying application running on the cluster. Each instance requires a unique name within the cluster. Each resource must be configured in a resource group. The RGM brings all resources in a group online and offline together on the same node. When the RGM brings a resource group online or offline, it invokes callback methods on the individual resources in the group.

The nodes on which a resource group is currently online are called its primary nodes, or its primaries. A resource group is mastered by each of its primaries. Each resource group has an associated Nodelist property, set by the cluster administrator, to identify all potential primaries or masters of the resource group.

For detailed information on the implementation and design of highly available IBM DB2 database environments on the Sun Cluster platform see the white paper entitled ″DB2 and High Availability on Sun Cluster 3.X″ which is available on the IBM Software Library web site (http://www.ibm.com/software/sw-library/).

VERITAS Cluster Server support:

If you plan to run your DB2 database solution on a Solaris Operating System cluster, you can use VERITAS Cluster Server to manager the cluster. VERITAS Cluster Server can manage a wide range of applications in heterogeneous environments; and it supports up to 32 node clusters in both storage area network (SAN) and traditional client/server environments.

Hardware Requirements

Following is a list of hardware currently supported by VERITAS Cluster Server:
- For server nodes:
  - Any SPARC/Solaris server from Sun Microsystems running Solaris 2.6 or later with a minimum of 128 MB RAM.
- For disk storage:
  - EMC Symmetrix, IBM Enterprise Storage Server®, HDS 7700 and 9xxx, Sun T3, Sun A5000, Sun A1000, Sun D1000 and any other disk storage supported by VCS 2.0 or later; your VERITAS representative can confirm which disk subsystems are supported or you can refer to VCS documentation.
  - Typical environments will require mirrored private disks (in each cluster node) for the DB2 binaries and shared disks between nodes for the DB2 data.
- For network interconnects:
  - For the public network connections, any network connection supporting IP-based addressing.
  - For the heartbeat connections (internal to the cluster), redundant heartbeat connections are required; this requirement can be met through the use of two additional Ethernet controllers per server or one additional Ethernet controller per server and the use of one shared GABdisk per cluster

Software Requirements

The following VERITAS software components are qualified configurations:
- VERITAS Volume Manager 3.2 or later, VERITAS File System 3.4 or later, VERITAS Cluster Server 2.0 or later.
- DB Edition for DB2 for Solaris 1.0 or later.

While VERITAS Cluster Server does not require a volume manager, the use of VERITAS Volume Manager is strongly recommended for ease of installation, configuration and management.

Failover

VERITAS Cluster Server is an availability clustering solution that manages the availability of application services, such as DB2 database, by enabling application failover. The state of each individual cluster node and its associated software services are regularly monitored. When a failure occurs that disrupts the application service (in this case, the DB2 database service), either VERITAS Cluster Server or the VCS HA-DB2 Agent, or both will detect the failure and automatically take steps to restore the service. The steps take to restore the service can include restarting the DB2 database system on the same node or moving DB2 database system to another node in the cluster and restarting it on that node. If an application needs to be migrated to a new node, VERITAS Cluster Server moves everything associated with the application (that is, network IP addresses, ownership of underlying storage) to the new node so that users will not be aware that the service is actually running on another node. They will still access the service using the same IP addresses, but those addresses will now point to a different cluster node.

When a failover occurs with VERITAS Cluster Server, users might or might not see a disruption in service. This will be based on the type of connection (stateful or stateless) that the client has with the application service. In application environments with stateful connections (like DB2 database), users might see a brief interruption in service and might need to reconnect after the failover has completed. In application environments with stateless connections (like NFS), users might see a brief delay in service but generally will not see a disruption and will not need to log back on.

By supporting an application as a service that can be automatically migrated between cluster nodes, VERITAS Cluster Server can not only reduce unplanned downtime, but can also shorten the duration of outages associated with planned downtime (for maintenance and upgrades). Failovers can also be initiated manually. If a hardware or operating system upgrade must be performed on a particular node, the DB2 database system can be migrated to another node in the cluster, the upgrade can be performed, and then the DB2 database system can be migrated back to the original node.

Applications recommended for use in these types of clustering environments should be crash tolerant. A crash tolerant application can recover from an unexpected crash while still maintaining the integrity of committed data. Crash tolerant applications are sometimes referred to as *cluster friendly* applications. DB2 database system is a crash tolerant application.

For information on how to decrease the amount of time it takes to perform a failover using a VERITAS CFS, CVM, and VCS solution, see the white paper entitled “DB2 Universal Database Version 8 and VERITAS Database Edition/HA for DB2”, which is available from the IBM Software Library web site (http://www.ibm.com/software/sw-library/).

Shared Storage

When used with the VCS HA-DB2 Agent, Veritas Cluster Server requires shared storage. Shared storage is storage that has a physical connection to multiple nodes in the cluster. Disk devices resident on shared storage can tolerate node failures since a physical path to the disk devices still exists through one or more alternate cluster nodes.

Through the control of VERITAS Cluster Server, cluster nodes can access shared storage through a logical construct called ″disk groups″. Disk groups represent a collection of logically defined storage devices whose ownership can be atomically migrated between nodes in a cluster. A disk group can only be imported to a single node at any given time. For example, if Disk Group A is imported to Node 1 and Node 1 fails, Disk Group A can be exported from the failed node and imported to a new node in the cluster. VERITAS Cluster Server can simultaneously control multiple disk groups within a single cluster.

In addition to allowing disk group definition, a volume manager can provide for redundant data configurations, using mirroring or RAID 5, on shared storage. VERITAS Cluster Server supports VERITAS Volume Manager and Solstice DiskSuite as logical volume managers. Combining shared storage with disk mirroring and striping can protect against both node failure and individual disk or controller failure.

VERITAS Cluster Server Global Atomic Broadcast(GAB) and Low Latency Transport (LLT)

An internode communication mechanism is required in cluster configurations so that nodes can exchange information concerning hardware and software status, keep track of cluster membership, and keep this information synchronized across all cluster nodes. The Global Atomic Broadcast (GAB) facility, running across a low latency transport (LLT), provides the high speed, low latency mechanism used by VERITAS Cluster Server to do this. GAB is loaded as a kernel module on each cluster node and provides an atomic broadcast mechanism that ensures that all nodes get status update information at the same time.

By leveraging kernel-to-kernel communication capabilities, LLT provides high speed, low latency transport for all information that needs to be exchanged and synchronized between cluster nodes. GAB runs on top of LLT. VERITAS Cluster Server does not use IP as a heartbeat mechanism, but offers two other more reliable options. GAB with LLT, can be configured to act as a heartbeat mechanism, or a GABdisk can be configured as a disk-based heartbeat. The heartbeat must run over redundant connections. These connections can either be two private Ethernet connections between cluster nodes, or one private Ethernet connection and one GABdisk connection. The use of two GABdisks is not a supported configuration since the exchange of cluster status between nodes requires a private Ethernet connection.

For more information about GAB or LLT, or how to configure them in VERITAS Cluster Server configurations, consult the VERITAS Cluster Server 2.0 User’s Guide for Solaris.

Bundled and Enterprise Agents

An agent is a program that is designed to manage the availability of a particular resource or application. When an agent is started, it obtains the necessary configuration information from VCS and then periodically monitors the resource or application and updates VCS with the status. In general, agents are used to bring resources online, take resources offline, or monitor resources and provide four types of services: start, stop, monitor and clean. Start and stop are used to bring resources online or offline, monitor is used to test a particular resource or application for its status, and clean is used in the recovery process.

A variety of bundled agents are included as part of VERITAS Cluster Server and are installed when VERITAS Cluster Server is installed. The bundled agents are VCS processes that manage predefined resource types commonly found in cluster configurations (that is, IP, mount, process and share), and they help to simplify cluster installation and configuration considerably. There are over 20 bundled agents with VERITAS Cluster Server.

Enterprise agents tend to focus on specific applications such as the DB2 database application. The VCS HA-DB2 Agent can be considered an Enterprise Agent, and it interfaces with VCS through the VCS Agent framework.

VCS Resources, Resource Types and Resource Groups

A resource type is an object definition used to define resources within a VCS cluster that will be monitored. A resource type includes the resource type name and a set of properties associated with that resource that are salient from a high availability point of view. A resource inherits the properties and values of its resource type, and resource names must be unique on a cluster-wide basis.

There are two types of resources: persistent and standard (non-persistent). Persistent resources are resources such as network interface controllers (NICs) that are monitored but are not brought online or taken offline by VCS. Standard resources are those whose online and offline status is controlled by VCS.

The lowest level object that is monitored is a resource, and there are various resource types (that is, share, mount). Each resource must be configured into a resource group, and VCS will bring all resources in a particular resource group online and offline together. To bring a resource group online or offline, VCS will invoke the start or stop methods for each of the resources in the group. There are two types of resource groups: failover and parallel. A highly available DB2 database configuration, regardless of whether it is partitioned database environment or not, will use failover resource groups.

A ″primary″ or ″master″ node is a node that can potentially host a resource. A resource group attribute called `systemlist` is used to specify which nodes within a cluster can be primaries for a particular resource group. In a two node cluster, usually both nodes are included in the `systemlist`, but in larger, multi-node clusters that might be hosting several highly available applications there might be a requirement to ensure that certain application services (defined by their resources at the lowest level) can never fail over to certain nodes.

Dependencies can be defined between resource groups, and VERITAS Cluster Server depends on this resource group dependency hierarchy in assessing the impact of various resource failures and in managing recovery. For example, if the resource group ClientApp1 can not be brought online unless the resource group DB2 has already been successfully started, resource group ClientApp1 is considered dependent on resource group DB2.

For detailed information on the implementation and design of highly available IBM DB2 database environments with the VERITAS Cluster Server, see the technote entitled ″DB2 UDB and High Availability with VERITAS Cluster Server″: http://www.ibm.com/support/docview.wss?rs=71&uid;=swg21045033

Synchronizing clocks in a partitioned database environment

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max_time_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database environment, DB2 uses the system clock and the virtual timestamp stored in the `SQLOGCTL.LFH` file on each machine as the basis for the time stamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set back, the clock for the logs cannot, and remains at the *same* advanced time until the system clock matches this time. The clocks are then in synchrony. The implication of this is that a short term system clock error on a database node can have a long lasting effect on the time stamps of database logs.

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 2005 when the year is 2003, and assume that the mistake is corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 2003 and November 7, 2005 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 2005, and the log clock remains at November 7, 2005 until the system clock matches this time. If you attempt to roll forward to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 2003.

Although DB2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

- The configurable values for this parameter range from 1 minute to 24 hours.
- When the first connection request is made to a non-catalog partition, the database partition server sends its time to the catalog partition for the database. The catalog partition then checks that the time on the database partition requesting the connection, and its own time are within the range specified by the *max_time_diff* parameter. If this range is exceeded, the connection is refused.
- An update transaction that involves more than two database partition servers in the database must verify that the clocks on the participating database partition servers are in synchrony before the update can be committed. If two or more database partition servers have a time difference that exceeds the limit allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated to other database partition servers.

Client/server timestamp conversion

This section explains the generation of timestamps in a client/server environment:

- If you specify a local time for a rollforward operation, all messages returned will also be in local time.

  Note: All times are converted on the server and (in partitioned database environments) on the catalog database partition.

- The timestamp string is converted to GMT on the server, so the time represents the server’s time zone, not the client’s. If the client is in a different time zone from the server, the server’s local time should be used.
- If the timestamp string is close to the time change due to daylight savings time, it is important to know whether the stop time is before or after the time change so that it is specified correctly.

Chapter 5. Administering and maintaining a highly available solution

Once you have created, configured, and started your DB2 database high availability solution running, there are ongoing activities you will have to perform. You need to monitor, maintain, and repair your database solution to keep it available to client applications.

As your database system runs, you need to monitor and respond to the following kinds of things:

1. Manage log files.

   Log files grow larger, require archiving; and some log files require copying or moving to be available for a restore operation.
2. Perform maintenance activities:
   - Installing software
   - Upgrading hardware
   - Reorganizing database tables
   - Database performance tuning
   - Database backup
3. Synchronize primary and secondary or standby databases so that failover works smoothly.
4. Identify and respond to unexpected failures in hardware or software.

Log file management

The DB2 database manager uses a number scheme to name log files. This naming strategy has implications for log file reuse and log sequences. Also, a DB2 database that has no client application connection uses a new log file when the next client application connects to that database server. These two aspects of DB2 Data Server database logging behavior affect the log file management choices you make.

Consider the following when managing database logs:

- The numbering scheme for archived logs starts with S0000000.LOG, and continues through S9999999.LOG, accommodating a potential maximum of 10 million log files. The database manager resets to S0000000.LOG if:
  - A database configuration file is changed to enable rollforward recovery
  - A database configuration file is changed to *disable* rollforward recovery
  - S9999999.LOG has been used.

  The DB2 database manager reuses log file names after restoring a database (with or without rollforward recovery). The database manager ensures that an incorrect log is not applied during rollforward recovery. If the DB2 database manager reuses a log file name after a restore operation, the new log files are archived to separate directories so that multiple log files with the same name can be archived. The location of the log files is recorded in the recovery history file so that they can be applied during rollforward recovery. You must ensure that the correct logs are available for rollforward recovery.

  When a rollforward operation completes successfully, the last log that was used is truncated, and logging begins with the next sequential log. Any log in the log path directory with a sequence number greater than the last log used for rollforward recovery is re-used. Any entries in the truncated log following the truncation point are overwritten with zeros. Ensure that you make a copy of the logs before invoking the rollforward utility. (You can invoke a user exit program to copy the logs to another location.)

- If a database has not been activated (by way of the ACTIVATE DATABASE command), the DB2 database manager truncates the current log file when all applications have disconnected from the database. The next time an application connects to the database, the DB2 database manager starts logging to a new log file. If many small log files are being produced on your system, you might want to consider using the ACTIVATE DATABASE command. This not only saves the overhead of having to initialize the database when applications connect, it also saves the overhead of having to allocate a large log file, truncate it, and then allocate a new large log file.
- An archived log can be associated with two or more different *log sequences* for a database, because log file names are reused (see Figure 8 on page 135). For example, if you want to recover Backup 2, there are two possible log sequences that could be used. If, during full database recovery, you roll forward to a point in time and stop before reaching the end of the logs, you have created a new log sequence. The two log sequences cannot be combined. If you have an online backup image that spans the first log sequence, you must use this log sequence to complete rollforward recovery.

  If you have created a new log sequence after recovery, any table space backup images on the old log sequence are invalid. This is usually recognized at restore time, but the restore utility fails to recognize a table space backup image on an old log sequence if a database restore operation is immediately followed by the table space restore operation. Until the database is actually rolled forward, the log sequence that is to be used is unknown. If the table space is on an old log sequence, it must be "caught" by the table space rollforward operation. A restore operation using an invalid backup image might complete successfully, but the table space rollforward operation for that table space will fail, and the table space will be left in restore pending state.

  For example, suppose that a table space-level backup operation, Backup 3, completes between `S0000013.LOG` and `S0000014.LOG` in the top log sequence (see Figure 8 on page 135). If you want to restore and roll forward using the database-level backup image, Backup 2, you will need to roll forward through `S0000012.LOG`. After this, you could continue to roll forward through either the top log sequence or the (newer) bottom log sequence. If you roll forward through the bottom log sequence, you will not be able to use the table space-level backup image, Backup 3, to perform table space restore and rollforward recovery.

  To complete a table space rollforward operation to the end of the logs using the table space-level backup image, Backup 3, you will have to restore the database-level backup image, Backup 2, and then roll forward using the top log sequence. Once the table space-level backup image, Backup 3, has been restored, you can initiate a rollforward operation to the end of the logs.

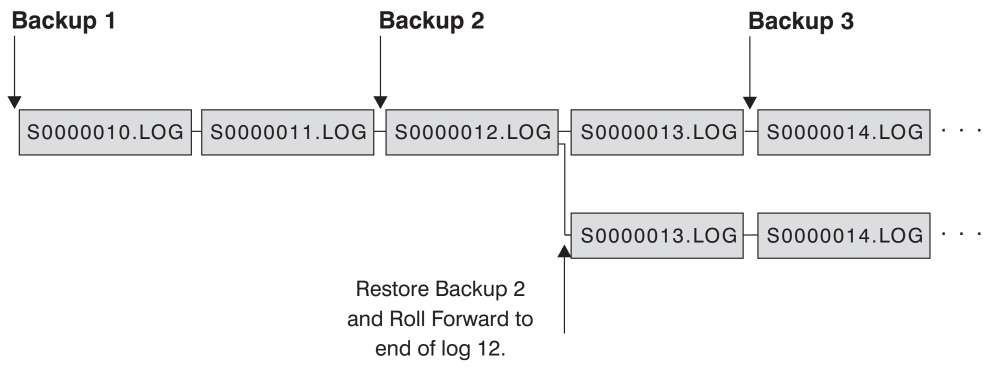


Figure 8. Re-using Log File Names

On demand log archive

IBM Data Server supports the closing (and, if enabled, the archiving) of the active log for a recoverable database at any time. This allows you to collect a complete set of log files up to a known point, and then to use these log files to update a standby database.

You can initiate on demand log archiving by invoking the ARCHIVE LOG command, or by calling the db2ArchiveLog API.

Log archiving using db2tapemgr

You can use the db2tapemgr utility to store archived log files to tape devices. The db2tapemgr utility copies log files from disk to the specified tape device, and updates the recovery history file with the new location of the copied log files.

Configuration

Set the database configuration parameter LOGARCHMETH1 to the location on disk of the log files you want to copy to tape. The db2tapemgr utility reads this LOGARCHMETH1 value to find the log files to copy. In a partitioned database environment, the LOGARCHMETH1 configuration parameter must be set on each database partition that contains log files to be copied.

The db2tapemgr utility does not use the LOGARCHMETH2 database configuration parameter.

STORE and DOUBLE STORE options

Issue the DB2TAPEMGR command with either the STORE or DOUBLE STORE option to transfer archived logs from disk to tape.

- The STORE option stores a range or all log files from the log archive directory to a specified tape device and deletes the files from disk.
- The DOUBLE STORE option scans the history file to see if logs have been stored to tape previously.
  - If a log has never been stored before, DB2TAPEMGR stores the log file to tape and but does not delete it from disk.
  - If a log has been stored before, DB2TAPEMGR stores the log file to tape and deletes it from disk.

Use DOUBLE STORE if you want to keep duplicate copies of your archived logs on tape and on disk, or if you want to store the same logs on two different tapes.

When you issue the DB2TAPEMGR command with either the STORE or DOUBLE STORE option, the db2tapemgr utility first scans the history file for entries where the LOGARCHMETH1 configuration parameter is set to disk. If it finds that any files that are supposed to be on disk, are not on disk, it issues a warning. If the db2tapemgr utility finds no log files to store, it stops the operation and issues a message to inform you that there is nothing to do.

RETRIEVE options

Issue the DB2TAPEMGR command with the RETRIEVE option to transfer files from tape to disk.

- Use the RETRIEVE ALL LOGS or LOGS n TO n option to retrieve all archived logs that meet your specified criteria and copy them to disk.
- Use the RETRIEVE FOR ROLLFORWARD TO POINT-IN-TIME option to retrieve all archived logs required to perform a rollforward operation and copy them to disk.
- Use the RETRIEVE HISTORY FILE option to retrieve the history file from tape and copy it to disk.

Behavior

- If the db2tapemgr utility finds log files on disk, it then reads the tape header to make sure that it can write the log files to the tape. It also updates the history for those files that are currently on tape. If the update fails, the operation stops and an error message is displayed.
- If the tape is writeable, the db2tapemgr utility copies the logs to tape. After the files have been copied, the log files are deleted from disk. Finally, the db2tapemgr utility copies the history file to tape and deletes it from disk.
- The db2tapemgr utility does not append log files to a tape. If a store operation does not fill the entire tape, then the unused space is wasted.
- The db2tapemgr utility stores log files only once to any given tape. This restriction exists to avoid any problems inherent to writing to tape media, such as stretching of the tape.
- In a partitioned database environment, the db2tapemgr utility only executes against one database partition at a time. You must run the appropriate command for each database partition, specifying the database partition number using the ON DBPARTITIONNUM option of the DB2TAPEMGR command. You must also ensure that each database partition has access to a tape device.

Examples

The following example shows how to use the DB2TAPEMGR command to store all log files from the primary archive log path for database sample on database partition number 0 to a tape device and remove them from the archive log path:

```
db2tapemgr db sample on dbpartitionnum 0 store on /dev/rmt0.1 all logs
```

The following example shows how to store the first 10 log files from the primary archive log path to a tape device and remove them from the archive log path:

```
db2tapemgr db sample on dbpartitionnum store on /dev/rmt0.1 10 logs
```

The following example shows how to store the first 10 log files from the primary archive log path to a tape device and then store the same log files to a second tape and remove them from the archive log path:

```
db2tapemgr db sample on dbpartitionnum double store on /dev/rmt0.1 10 logs
db2tapemgr db sample on dbpartitionnum double store on /dev/rmt1.1 10 logs
```

The following example shows how to retrieve all log files from a tape to a directory:

```
db2tapemgr db sample on dbpartitionnum retrieve all logs from /dev/rmt1.1
      to /home/dbuser/archived_logs
```

Automating log file archiving and retrieval with user exit programs

You can automate log file archiving and retrieval by creating a user exit program that the DB2 database manager calls to carry out the archiving or retrieval operation.

When the DB2 database manager invokes your user exit program, the following happens:

- the database manager passes control to the user exit program;
- the database manager passes parameters to the user exit program; and
- on completion, the use exit program passes a return code back to the database manager.

Configuration

Before invoking a user exit program for log file archiving or retrieval, ensure that the logarchmeth1 database configuration parameter has been set to USEREXIT. This also enables your database for rollforward recovery.

User exit program requirements

- The executable file for your user exit program must be called db2uext2.
- User exit programs must copy log files from the active log path to the archive log path, not move them. Do not remove log files from the active log path. If you remove log files from the active log path, your DB2 database might not be able to successfully recover in the event of a failure.

  DB2 database requires the log files to be in the active log path during recovery. The DB2 database server will remove archived log files from the active log path when these log files are no longer needed for recovery.
- User exit programs must handle error conditions. Your user exit program must handle errors because the DB2 database manager can only handle a limited set of return conditions.

  See “User exit error handling” on page 139.
-

Each DB2 database manager instance can only invoke one user exit program. Because the database manager instance can only invoke one user exit program, you must design your user exit program with a section for each operation it might have to perform.

Sample user exit programs

Sample user exit programs are provided for all supported platforms. You can modify these programs to suit your particular requirements. The sample programs are well commented with information that will help you to use them most effectively.

You should be aware that user exit programs must *copy* log files from the active log path to the archive log path. Do not remove log files from the active log path. (This could cause problems during database recovery.) DB2 removes archived log files from the active log path when these log files are no longer needed for recovery.

Following is a description of the sample user exit programs that are shipped with DB2 Data Server.

- **UNIX based systems**

  The user exit sample programs for DB2 Data Serve for UNIX based systems are found in the sqllib/samples/c subdirectory. Although the samples provided are coded in C, your user exit program can be written in a different programming language.

  Your user exit program must be an executable file whose name is db2uext2.

  There are four sample user exit programs for UNIX based systems:

  – db2uext2.ctsm

    This sample uses Tivoli Storage Manager to archive and retrieve database log files.

  – db2uext2.ctape

    This sample uses tape media to archive and retrieve database log files .

  – db2uext2.cdisk

    This sample uses the operating system COPY command and disk media to archive and retrieve database log files.

  – db2uxt2.cxbsa

    This sample works with the XBSA Draft 0.8 published by the X/Open group. It can be used to archive and retrieve database log files. This sample is only supported on AIX.

- **Windows operating systems**

  The user exit sample programs for DB2 Data Server for Windows operating systems are found in the sqllib\samples\c subdirectory. Although the samples provided are coded in C, your user exit program can be written in a different programming language.

  Your user exit program must be an executable file whose name is db2uext2.

  There are two sample user exit programs for Windows operating systems:

  – db2uext2.ctsm

    This sample uses Tivoli Storage Manager to archive and retrieve database log files.

  – db2uext2.cdisk

    This sample uses the operating system COPY command and disk media to archive and retrieve database log files.

User exit program calling format

When the DB2 database manager calls a user exit program, it passes a set of parameters (of data type CHAR) to the program.

Command syntax

```
db2uext2 -OS<os> -RL<db2rel> -RQ<request> -DB<dbname>
-NN<nodenum> -LP<logpath> -LN<logname> -AP<tsmpasswd>
-SP<startpage> -LS<logsize>
```

os Specifies the platform on which the instance is running. Valid values are: `AIX`, `Solaris`, `HP-UX`, `SCO`, `Linux`, and `NT`.

db2rel Specifies the DB2 release level. For example, `SQL07020`.

request
Specifies a request type. Valid values are: `ARCHIVE` and `RETRIEVE`.

dbname
Specifies a database name.

nodenum
Specifies the local node number, such as `5`, for example.

logpath
Specifies the fully qualified path to the log files. The path must contain the trailing path separator. For example, `/u/database/log/path/`, or `d:\logpath\`.

logname
Specifies the name of the log file that is to be archived or retrieved, such as `S0000123.LOG`, for example.

tsmpasswd
Specifies the TSM password. (If a value for the database configuration parameter *tsm_password* has previously been specified, that value is passed to the user exit program.)

startpage
Specifies the number of 4-KB offset pages of the device at which the log extent starts.

logsize
Specifies the size of the log extent, in 4-KB pages. This parameter is only valid if a raw device is used for logging.

User exit error handling

If you create a user exit program to automate log file archiving and retrieval, your user exit program passes return codes to the DB2 database manager that invoked the user exit program. The DB2 database manager can only handle a limited list of specific error codes. However, your user exit program might encounter many different kinds of error conditions, such as operating system errors. Your user exit program must map the error conditions it encounters to error codes that the database manager can handle.

Table 8 on page 140 shows the codes that can be returned by a user exit program, and describes how these codes are interpreted by the database manager. If a return code is not listed in the table, it is treated as if its value were 32.

Table 8. User Exit Program Return Codes. Applies to archiving and retrieval operations only.

Return Code	Explanation
0	Successful.
4	Temporary resource error encountered.[a]
8	Operator intervention is required.[a]
12	Hardware error.[b]
16	Error with the user exit program or a software function used by the program.[b]
20	Error with one or more of the parameters passed to the user exit program. Verify that the user exit program is correctly processing the specified parameters.[b]
24	The user exit program was not found. [b]
28	Error caused by an input/output (I/O) failure, or by the operating system.[b]
32	The user exit program was terminated by the user.[b]
255	Error caused by the user exit program not being able to load the library file for the executable.[c]

[a] For archiving or retrieval requests, a return code of 4 or 8 causes a retry in five minutes. If the user exit program continues to return 4 or 8 on retrieve requests for the same log file, DB2 will continue to retry until successful. (This applies to rollforward operations, or calls to the db2ReadLog API, which is used by the replication utility.)

[b] User exit requests are suspended for five minutes. During this time, all requests are ignored, including the request that caused the error condition. Following this five-minute suspension, the next request is processed. If this request is processed without error, processing of new user exit requests continues, and DB2 reissues the archive request that failed or was suspended previously. If a return code greater than 8 is generated during the retry, requests are suspended for an additional five minutes. The five-minute suspensions continue until the problem is corrected, or the database is stopped and restarted. Once all applications have disconnected from the database, DB2 issues an archive request for any log file that might not have been successfully archived previously. If the user exit program fails to archive log files, your disk might become filled with log files, and performance might be degraded. Once the disk becomes full, the database manager will not accept further application requests for database updates. If the user exit program was called to retrieve log files, rollforward recovery is suspended, but not stopped, unless the ROLLFORWARD STOP option was specified. If the STOP option was not specified, you can correct the problem and resume recovery.

[c] If the user exit program returns error code 255, it is likely that the program cannot load the library file for the executable. To verify this, manually invoke the user exit program. More information is displayed.

Note: During archiving and retrieval operations, an alert message is issued for all return codes except 0, and 4. The alert message contains the return code from the user exit program, and a copy of the input parameters that were provided to the user exit program.

Log file allocation and removal

Log files in the database log directory are never removed if they might be required for crash recovery. A log file which is required for crash recovery is called an active log. A log file which is not required for crash recovery is called an archived log.

If you have enabled infinite logging, log files will be deleted once they have been successfully archived. When the logarchmeth1 database configuration parameter is not set to OFF, a full log file becomes a candidate for removal only after it is no longer required for crash recovery.

The process of allocating new log files and removing old log files is dependent on the settings of the *logarchmeth1* database configuration parameter:

Logarchmeth1 **and** *Logarchmeth2* **are set to `OFF`**

Circular logging will be used. Rollforward recovery is not supported with circular logging, while crash recovery is.

During circular logging, new log files, other than secondary logs, are not generated and old log files are not deleted. Log files are handled in a circular fashion. That is, when the last log file is full, DB2 begins writing to the first log file.

A log full situation can occur if all of the log files are active and the circular logging process cannot wrap to the first log file. Secondary log files are created when all the primary log files are active and full. Secondary log files are deleted when the database is deactivated or when the space they are using is required for the active log files.

Logarchmeth1 **is set to `LOGRETAIN`**

Archive logging is used. The database is a recoverable database. Both rollforward recovery and crash recovery are enabled. After you archive the log files, you must delete them from the active log path so that the disk space can be reused for new log files. Each time a log file becomes full, DB2 begins writing records to another log file, and (if the maximum number of primary and secondary logs has not been reached) creates a new log file.

Logarchmeth1 **is set to a value other than `OFF` or `LOGRETAIN`**

Archive logging is used. The database is a recoverable database. Both rollforward recovery and crash recovery are enabled. When a log file becomes full, it is automatically archived.

Log files are usually not deleted. Instead, when a new log file is required and one is not available, an archived log file is renamed and used again. An archived log file, is not deleted or renamed once it has been closed and copied to the log archive directory. DB2 waits until a new log file is needed and then renames the oldest archived log. A log file that has been moved to the database directory during recovery is removed during the recovery process when it is no longer needed. Until DB2 runs out of log space, you will see old log files in the database directory.

If an error occurs when log files are being archived, archiving is suspended for the amount of time specified by the ARCHRETRYDELAY database configuration parameter. You can also use the NUMARCHRETRY database configuration parameter to specify the number of times that DB2 is to try archiving a log file to the primary or secondary archive directory before it tries to archive log files to the failover directory (specified by the FAILARCHPATH database configuration parameter). NUMARCHRETRY is only used if the FAILARCHPATH database configuration parameter is set. If NUMARCHRETRY is set to 0, DB2 will continuously retry archiving from the primary or the secondary log path.

The easiest way to remove old log files is to restart the database. Once the database is restarted, only new log files and log files that the user exit program failed to archive will be found in the database directory.

When a database is restarted, the minimum number of logs in the database log directory will equal the number of primary logs which can be configured using the *logprimary* database configuration parameter. It is possible for more than the number of primary logs to be found in the log directory. This can occur if the number of empty logs in the log directory at the time the database was shut down, is greater than the value of the *logprimary* configuration parameter at the time the database is restarted. This will happen if the value of the *logprimary* configuration parameter is changed between the database being shut down and restarted, or if secondary logs are allocated and never used.

When a database is restarted, if the number of empty logs is less than the number of primary logs specified by the *logprimary* configuration parameter, additional log files will be allocated to make up the difference. If there are more empty logs than primary logs available in the database directory, the database can be restarted with as many available empty logs as are found in the database directory. After database shutdown, secondary log files that have been created will remain in the active log path when the database is restarted.

Including log files with a backup image

When performing an online backup operation, you can specify that the log files required to restore and recover a database are included in the backup image. This means that if you need to ship backup images to a disaster recovery site, you do not have to send the log files separately or package them together yourself. Further, you do not have to decide which log files are required to guarantee the consistency of an online backup. This provides some protection against the deletion of log files required for successful recovery.

To make use of this feature specify the INCLUDE LOGS option of the BACKUP DATABASE command. When you specify this option, the backup utility will truncate the currently active log file and copy the necessary set of log extents into the backup image.

To restore the log files from a backup image, use the LOGTARGET option of the RESTORE DATABASE command and specify a fully qualified path that exists on the DB2 server. The restore database utility will then write the log files from the image to the target path. If a log file with the same name already exists in the target path, the restore operation will fail and an error will be returned. If the LOGTARGET option is not specified, no log files will be restored from the backup image.

If the LOGTARGET option is specified and the backup image does not include any log files, an error will be returned before an attempt is made to restore any table space data. The restore operation will also fail if an invalid or read-only path is specified. During a database or table space restore where the LOGTARGET option is specified, if one or more log files cannot be extracted, the restore operation fails and an error is returned.

You can also choose to restore only the log files saved in the backup image. To do this, specify the LOGS option with the LOGTARGET option of the RESTORE DATABASE command. If the restore operation encounters any problems when restoring log files in this mode, the restore operation fails and an error is returned.

During an automatic incremental restore operation, only the logs included in the target image of the restore operation will be retrieved from the backup image. Any logs that are included in intermediate images referenced during the incremental restore process will not be extracted from those backup images. During a manual incremental restore, if you specify a log target directory when restoring a backup image that includes log files, the log files in that backup image will be restored.

If you roll a database forward that was restored from an online backup image that includes log files, you might encounter error SQL1268N, which indicates roll-forward recovery has stopped due to an error received when retrieving a log. This error is generated when the target system to which you are attempting to restore the backup image does not have access to the facility used by the source system to archive its transaction logs.

If you specify the INCLUDE LOGS option of the BACKUP DATABASE command when you back up a database, then subsequently perform a restore operation and a roll-forward operation that use that backup image, DB2 will still search for additional transaction logs when rolling the database forward, even though the backup image includes logs. It is standard rollforward behaviour to continue to search for additional transaction logs until no more logs are found. It is possible to have more than one log file with the same timestamp. Consequently, DB2 does not stop as soon as it finds the first timestamp that matches the point-in-time to which you are rolling forward the database as there might be other log files that also have that timestamp. Instead, DB2 continues to look at the transaction log until it finds a timestamp greater than the point-in-time specified.

When no additional logs can be found, the rollforward operation ends successfully. However, if there is an error while searching for additional transaction log files, error SQL1268N is returned. Error SQL1268N can occur because during the initial restore, certain database configuration parameters were reset or overwritten. Three of these database configuration parameters are the TSM parameters, TSM_NODENAME, TSM_OWNER and TSM_PASSWORD. They are all reset to NULL. To rollforward to the end of logs, you need to reset these database configuration parameters to correspond to the source system prior to the rollforward operation. Alternatively, you can specify the NORETRIEVE option when you issue the ROLLFORWARD DATABASE command. This will prevent the DB2 database system from trying to obtain potentially missing transaction logs elsewhere.

Note:
1. This feature is not supported for offline backups.
2. When logs are included in an online backup image, the resulting image cannot be restored on releases of DB2 database prior to Version 8.2.

Preventing the accidental loss of log files

In situations where you need to drop a database or perform a point-in-time rollforward recovery, it is possible to lose log files that might be required for future recovery operations. In these cases, it is important to make copies of all the logs in the current database log path directory.

Consider the following scenarios:

-

  If you plan to drop a database prior to a restore operation, you need to save the log files in the active log path before issuing the DROP DATABASE command. After the database has been restored, these log files might be required for rollforward recovery because some of them might not have been archived before the database was dropped. Normally, you are not required to drop a database prior to issuing the RESTORE command. However, you might have to drop the database (or drop the database on one database partition by specifying the AT® NODE option of DROP DATABASE command), because it has been damaged to the extent that the RESTORE command fails. You might also decide to drop a database prior to the restore operation to give yourself a fresh start.

- If you are rolling a database forward to a specific point in time, log data after the time stamp you specify will be overwritten. If, after you have completed the point-in-time rollforward operation and reconnected to the database, you determine that you actually needed to roll the database forward to a later point in time, you will not be able to because the logs will already have been overwritten. It is possible that the original set of log files might have been archived; however, DB2 might be calling a user exit program to automatically archive the newly generated log files. Depending on how the user exit program is written, this could cause the original set of log files in the archive log directory to be overwritten. Even if both the original and new set of log files exist in the archive log directory (as different versions of the same files), you might have to determine which set of logs should be used for future recovery operations.

Minimizing the impact of maintenance on availability

You will have to perform maintenance on your DB2 database solution such as: software or hardware upgrades; database performance tuning; database backups; statistics collection; and monitoring for business purposes. Minimizing the impact that performing that maintenance has on the availability of your solution involves careful scheduling of offline maintenance, and using DB2 features and functionality that reduce the availability impact of online maintenance.

Before you can use the following steps to minimize the impact of maintenance on the availability of your DB2 database solution, you must:

- configure automatic maintenance; and
- install the High Availability Disaster Recovery (HADR) feature.

1. Allow automatic maintenance to do your maintenance for you.

   DB2 database can automate many database maintenance activities. Once the automatic maintenance has been configured, the maintenance will happen without you taking any additional steps to perform that maintenance.

2. Use a DB2 High Availability Disaster Recovery (HADR) rolling upgrade to minimize the impact of other maintenance activities.

   If you are upgrading software or hardware, or if you are modifying some database manager configuration parameters, the HADR feature enables you to accomplish those changes with minimal interruption of availability. This seamless change enabled by HADR is called a rolling upgrade.

   Some maintenance activities require you to shut down a database before performing the maintenance, even in the HADR environment. Under some conditions, the procedure for shutting down an HADR database is a little different than the procedure for shutting down a standard database: if an HADR database is started by a client application connecting to it, you must use the DEACTIVATE DATABASE command.

Stopping DB2 High Availability Disaster Recovery (HADR)

If you are using the DB2 High Availability Disaster Recovery (HADR) feature, performing maintenance on the two DB2 database systems, the primary and the standby, can be more complicated than performing maintenance on one standalone database server. If you need to stop HADR to perform maintenance, use the STOP HADR command to stop HADR functionality. If you are performing maintenance on the standby database system only, you only need to stop HADR on the standby database. To stop using HADR completely, you can stop HADR on both databases.

Warning: If you want to stop the specified database but you still want it to maintain its role as either an HADR primary or standby database, do not issue the STOP HADR command. If you issue the STOP HADR command the database will become a standard database and might require reinitialization in order to resume operations as an HADR database. Instead, issue the DEACTIVATE DATABASE command.

You can issue the STOP HADR command against a primary or a standby database only. If you issue this command against a standard database an error will be returned.

You can stop HADR by using the command line processor (CLP), the Manage High Availability Disaster Recovery (HADR) window in the Control Center, or the db2HADRStop application programming interface (API).

To use the CLP to stop HADR operations on the primary or standby database, issue the STOP HADR command on the database where you want to stop HADR operations.

In the following example, HADR operations are stopped on database SOCKS:

```
STOP HADR ON DATABASE SOCKS
```

If you issue this command against an inactive primary database, the database switches to a standard database and remains offline.

If you issue this command against an inactive standby database the database switches to a standard database, is placed in rollforward pending state, and remains offline.

If you issue this command on an active primary database, logs stop being shipped to the standby database and all HADR engine dispatchable units (EDUs) are shut down on the primary database. The database switches to a standard database and remains online. Transaction processing can continue. You can issue the START HADR command with the AS PRIMARY option to switch the role of the database back to primary database.

If you issue this command on an active standby database, an error message is returned, indicating that you must deactivate the standby database before attempting to convert it to a standard database.

To open the Stop HADR window:

1. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.
2. Click Stop HADR. The Stop HADR window opens.

3. If you want to stop HADR on one database only, clear the check box for the other database.
4. If only one database is started (either the primary database or the standby database), the name of that database is displayed in the Stop HADR window.
5. Click OK. The window closes. A progress indicator might open to indicate when the command is running. When it completes, you will get a notification indicating whether or not it is successful.

   Additional information is provided through the contextual help facility within the Control Center.

Database activation and deactivation in a DB2 High Availability Disaster Recovery (HADR) environment

If a standard database is started by a client connection, the database is shut down when the last client disconnects. If an HADR primary database is started by a client connection, it is equivalent to starting the database by using the ACTIVATE DATABASE command. To shut down an HADR primary database that was started by a client connection, you need to explicitly issue the DEACTIVATE DATABASE command.

On a standard database in rollforward pending state, the ACTIVATE DATABASE and DEACTIVATE DATABASE commands are not applicable. You can only continue rollforward, stop rollforward, or use START HADR start the database as an HADR standby database. Once a database is started as an HADR standby, you can use the ACTIVATE DATABASE and DEACTIVATE DATABASE commands to start and stop the database.

Activate a primary database using the following methods:
- client connection
- ACTIVATE DATABASE command
- START HADR command with the AS PRIMARY option

Deactivate a primary database using the following methods:
- DEACTIVATE DATABASE command
- db2stop command with the FORCE option

Activate a standby database using the following methods:
- ACTIVATE DATABASE command
- START HADR command with the AS STANDBY option

Deactivate a standby database using the following methods:
- DEACTIVATE DATABASE command
- db2stop command with the FORCE option

Performing a rolling upgrade in a DB2 High Availability Disaster Recovery (HADR) environment

Use this procedure in a high availability disaster recovery (HADR) environment when you upgrade software or hardware, update your DB2 database system, or when you make changes to database configuration parameters. This procedure keeps database service available throughout the upgrade process, with only a momentary service interruption when processing is switched from one database to the other. Because HADR performs optimally when both the primary and standby databases are on identical systems, you should apply changes to both systems as quickly as possible.

Note: All DB2 database system fix packs and upgrades should be implemented in a test environment before being applied to your production system.

The HADR pair should be in peer state before starting the rolling upgrade.

This procedure will not work to upgrade from an earlier to a later version of a DB2 database system; for example, you cannot use this procedure to upgrade from a version 8 to a version 9 database system. You can use this procedure to perform a rolling update on your database system from one modification level to another only, for example by applying a fix pack.

This procedure will not work if you update the DB2 HADR configuration parameters. Updates to HADR configuration parameters should be made separately. Because HADR requires the parameters on the primary and standby to be the same, this might require both the primary and standby databases to be deactivated and updated at the same time.

To perform a rolling upgrade in an HADR environment:

1. Upgrade the system where the standby database resides:
   a. Use the DEACTIVATE DATABASE command to shut down the standby database.
   b. If necessary, shut down the instance on the standby database.
   c. Make changes to one or more of the following: the software, the hardware, or the DB2 configuration parameters.

      Note: You cannot change any HADR configuration parameters when performing a rolling upgrade.
   d. If necessary, restart the instance on the standby database.
   e. Use the ACTIVATE DATABASE command to restart the standby database.
   f. Ensure that the standby database enters peer state. Use the GET SNAPSHOT command to check this.
2. Switch the roles of the primary and standby databases:
   a. Issue the TAKEOVER HADR command on the standby database.
   b. Direct clients to the new primary database. This can be done using automatic client reroute.

      Note: Because the standby database takes over as the primary database, the new primary database is now upgraded. If you are applying a DB2 database system fix pack, the TAKEOVER HADR command will change the role of the original primary database to standby database. However, the command will not let the new standby database connect to the newly updated primary database. Because the new standby database uses an older version of the DB2 database system, it might not understand the new log records generated by the updated primary database, and it will be shut down. In order for the new standby database to reconnect with the new primary database (that is, for the HADR pair to reform), the new standby database must also be updated.
3. Upgrade original primary database (which is now the standby database) using the same procedure as in Step 1 above. When you have done this, both

databases are upgraded and connected to each other in HADR peer state. The HADR system provides full database service and full high availability protection.

4. Optional. To return to your original configuration, switch the roles of the primary and standby database as in step 2.

Using a split mirror to clone a database

Use the following procedure to create a clone database. Although you can write to clone databases, they are generally used for read-only activities such as running reports.

You cannot back up a cloned database, restore the backup image on the original system, or roll forward through log files produced on the original system. You can use the AS SNAPSHOT option, but this provides only an instantaneous copy of the database at that time when the I/O is suspended; any other outstanding uncommitted work will be rolled back after the db2inidb command is executed on the clone.

To clone a database, follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
   While the database is suspended, you should not be running other utilities or tools. You should only be making a copy of the database.
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.

   Note: Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. Catalog the mirrored database on the secondary system.

   Note: By default, a mirrored database cannot exist on the same system as the primary database. It must be located on a secondary system that has the same directory structure and uses the same instance name as the primary database. If the mirrored database must exist on the same system as the primary database, you can use the db2relocatedb utility or the RELOCATE USING option of the db2inidb command to accomplish this.
5. Start the database instance on the secondary system:
   ```
   db2start
   ```
6. Initialize the mirrored database on the secondary system:
   ```
   db2inidb database_alias as snapshot
   ```
   If required, specify the RELOCATE USING option of the db2inidb command to relocate the clone database:
   ```
    db2inidb database_alias as snapshot relocate using relocatedbcfg.txt
   ```
   where the relocatedbcfg.txt file contains the information required to relocate the database.

   Notes:

a. This command will roll back transactions that are in flight when the split occurs, and start a new log chain sequence so that any logs from the primary database cannot be replayed on the cloned database.
b. The database directory (including the volume directory), the log directory, and the container directories must be moved to the desired location before you use the RELOCATE USING option.

Synchronizing the primary and standby databases

One high availability strategy is to have a primary database and a secondary or standby database to take over operations if the primary database fails. If the standby database must take over database operations for a failed primary database, it must contain exactly the same data, know about all inflight transactions, and otherwise continue database processing exactly the same way as the primary database server would, if it had not failed. The ongoing process of updating the standby database so that it is a copy of the primary database is called synchronization.

Before you can synchronize the primary and standby databases you must:

- Create and configure the primary and standby databases.
- Configure communications between the primary and standby databases.
-

  Choose a synchronization strategy (for example, log shipping, log mirroring, suspended I/O and disk mirroring, or HADR.)

  There are several strategies for keeping the primary database server and the standby database server synchronized:
  - shipping logs from the primary database to the standby database and rolling them forward on the standby database;
  - writing database logs to both the primary and standby databases at the same time, known as log mirroring;
  - using suspended I/O support with disk mirroring to periodically taking a copy of the primary database, splitting th mirror and initializing the copy as a new standby database server; and
  - using a availability feature such as the DB2 High Availability Disaster Recovery (HADR) feature to keep the primary and standby database synchronized.

1. If you are using logs to synchronize the primary database and the secondary or standby database, configure DB2 database to perform the required log management for you. For example, if you want DB2 database to mirror the logs, set the MIRRORLOGPATH configuration parameter to the location where you want the second copy of the logs to be saved.
2. If you are using DB2 database suspended I/O functionality to split a disk mirror of the primary database, you must do the following:
   a. Initialize the disk mirroring for the primary database.
   b. When you need to split the mirror of the primary database, follow the instructions in the topic "Using a split mirror as a standby database."
3. If you are using the HADR feature to manage synchronizing the primary and standby databases, configure DB2 database for HADR, and allow DB2 database to synchronize the primary and standby databases for you.

DB2 High Availability Disaster Recovery (HADR) replicated operations

DB2 High Availability Disaster Recovery (HADR) uses database logs to replicate data from the primary database to the standby database. Some activities can cause the standby database to fall behind the primary database as logs are replayed on the standby database. Some activities are so heavily logged that the large amount of log files they generate can cause storage problems. Although replicating data to the standby database using logs is the core of availability strategies, logging itself can potentially have a negative impact on the availability of your solution. Design you maintenance strategy wisely, configure your system to minimize the negative impact of logging, and allow logging to protect your transaction data.

In high availability disaster recovery (HADR), the following operations are replicated from the primary to the standby database:

- Data definition language (DDL)
- Data manipulation language (DML)
- Buffer pool operations
- Table space operations
- Online reorganization
- Offline reorganization
- Metadata for stored procedures and user defined functions (UDF) (but not the related object or library files)

During an online reorganization, all operations are logged in detail. As a result, HADR can replicate the operation without the standby database falling further behind than it would for more typical database updates. However, this behavior can potentially have a large impact on the system because of the large number of log records generated.

While offline reorganizations are not logged as extensively as online reorganizations, operations are typically logged per hundreds or thousands of affected rows. This means that the standby database could fall behind because it waits for each log record and then replays many updates at once. If the offline reorganization is non-clustered, a single log record is generated after the entire reorganization operation. This mode has the greatest impact on the ability of the standby database to keep up with the primary database. The standby database will perform the entire reorganization after it receives the log record from the primary database.

HADR does not replicate stored procedure and UDF object and library files. You must create the files on identical paths on both the primary and standby databases. If the standby database cannot find the referenced object or library file, the stored procedure or UDF invocation will fail on the standby database.

DB2 High Availability Disaster Recovery (HADR) non-replicated operations

DB2 High Availability Disaster Recovery (HADR) uses database logs to replicate data from the primary database to the standby database. Non-logged operations are allowed on the primary database, but not replicated to the standby database. If you want non-logged operations, such as updates to the history file, to be reflected in the standby database, you must take extra steps to cause this to happen.

The following are examples of cases in which operations on the primary database are not replicated to the standby database:

- Tables created with the NOT LOGGED INITIALLY option specified are not replicated. Attempts to access such tables after an HADR standby database takes over as the primary database will result in an error.
- BLOBs and CLOBs that are larger than 1GB cannot be logged, so they cannot be replicated. Non-logged BLOBs and CLOBs are not replicated. However, the space for them will be allocated on the standby database. The data for the LOB column will be binary zeroes. All logged BLOBs and CLOBs are replicated.
- Updates to database configuration using the UPDATE DATABASE CONFIGURATION and UPDATE DATABASE MANAGER CONFIGURATION commands are not replicated.
- Database configuration and database manager configuration parameters are not replicated.
- For user-defined functions (UDFs), changes to objects external to the database (such as related objects and library files) are not replicated. They will need to be setup on the standby via other means.
- The recovery history file (db2rhist.asc), and changes to it, are not automatically shipped from the primary database to the standby database.

  You can place an initial copy of the history file (obtained from the backup image of the primary) on the standby database by issuing the RESTORE DATABASE command with the REPLACE HISTORY FILE option:

  ```
  RESTORE DB KELLY REPLACE HISTORY FILE
  ```

  After HADR is initialized and subsequent backup activities take place on the primary database, the history file on the standby database will become out of date. However, a copy of the history file is stored in each backup image. You can update the history file on the standby by extracting the history file from a backup image using the following command:

  ```
  RESTORE DB KELLY HISTORY FILE
  ```

  Do not use regular operating system commands to copy the history file in the database directory from the primary database to the standby database. The history file can become corrupted if the primary is updating the files when the copy is made.

  If a takeover operation occurs and the standby database has an up-to-date history file, backup and restore operations on the new primary will generate new records in the history file and blend seamlessly with the records generated on the original primary. If the history file is out of date or has missing entries, an automatic incremental restore might not be possible; instead, a manual incremental restore operation will be required.

DB2 High Availability Disaster Recovery (HADR) standby database states

At any time, the standby database is in one of five states: local catchup; remote catchup pending; remote catchup; peer; and disconnected peer. The state that the standby database is in determines what operations it is capable of performing. You can use the GET SNAPSHOT command to see what state your standby database is in.

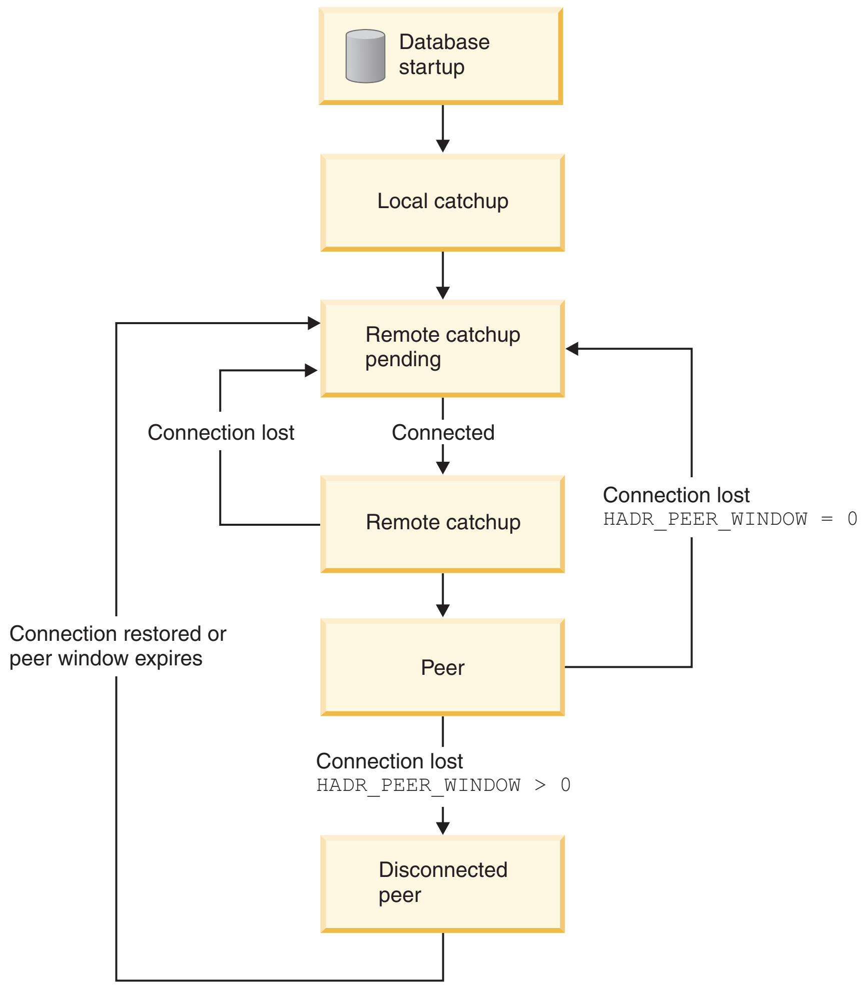

Database startup
Local catchup
Remote catchup pending
Connection lost
Connected
Remote catchup
Connection lost HADR_PEER_WINDOW = 0
Connection restored or peer window expires
Peer
Connection lost HADR_PEER_WINDOW > 0
Disconnected peer

Figure 9. States of the standby database

Database startup, local catchup, and remote catchup pending

With the high availability disaster recovery (HADR) feature, when the standby database is started, it enters local catchup state and attempts to read the log files in its local log path. If it does not find a log file in the local log path and a log archiving method has been specified, the log file is retrieved using the specified method. After the log files are read, they are replayed on the standby database. During this time, a connection to the primary database is not required; however, if a connection does not exist, the standby database tries to connect to the primary database. When the end of local log files is reached, the standby database enters remote catchup pending state.

If local log files become available after the standby database enters remote catchup pending state, you can shut down the standby database and restart it to cause it to re-enter local catchup state. You might do this if local access to such log files on the standby database is more efficient than allowing HADR to copy the files over the network from the primary database.

Remote catchup pending, remote catchup, peer

The standby database remains in remote catchup pending state until a connection to the primary database is established, at which time the standby database enters remote catchup state. During this time, the primary database reads log data from its log path or by way of a log archiving method and sends the log files to the standby database. The standby database receives and replays the log data. The primary and standby databases enter peer state when the standby database receives all of the log files that are on the disk of the primary database machine.

When in peer state, log pages are shipped to the standby database whenever the primary database flushes a log page to disk. The log pages are written to the local log files on the standby database to ensure that the primary and standby databases have identical log file sequences. The log pages can then be replayed on the standby database.

If the connection between the primary and standby databases is lost when the databases are in remote catchup state, the standby database will enter remote catchup pending state. If the connection between the primary and standby databases is lost when the databases are in peer state, and if the HADR_PEER_WINDOW database configuration parameter is not set (or set to zero) then the standby database will enter remote catchup pending state. However, if the connection between the primary and standby databases is lost when the databases are in peer state, and if the HADR_PEER_WINDOW database configuration parameter is set to a non-zero value, then the standby database enters disconnected peer state.

Disconnected peer

If you configure the database configuration parameter HADR_PEER_WINDOW to a time value that is greater than zero, then if the primary database loses connection with the standby database, then the primary database will continue to behave as though the primary and standby databases were in peer state for the configured amount of time. When the primary database and standby database are disconnected, but behaving as though in peer state, this state is called disconnected peer. The period of time for which the primary database remains in disconnected peer state after losing connection with the standby database is called the peer window. When the connection to the standby database is restored or the peer window expires, the standby database leaves the disconnected peer state.

The advantage of configuring a peer window is a lower risk of transaction loss during multiple or cascading failures. Without the peer window, when the primary database loses connection with the standby database, the primary database moves out of peer state. When the primary database is disconnected, it processes transactions independent of the standby database. If a failure occurs on the primary database while it is not in peer state like this, then transactions could be lost because they have not been replicated on the standby database. With the peer window configured, the primary database will not consider a transaction committed until the primary database has received acknowledgement from the standby database that the logs have been written to main memory on the standby system, or that the logs have been written to log files on the standby database (depending on the HADR synchronization mode.)

The disadvantage of configuring a peer window is that transactions on the primary database will take longer or even time out while the primary database is in the peer window waiting for the connection with the standby database to be restored or for the peer window to expire.

You can determine the peer window size, which is the value of the HADR_PEER_WINDOW database configuration parameter, using the GET SNAPSHOT command or the db2pd utility with the -hadr parameter.

Implications and restrictions of these standby database states for synchronizing the primary and standby databases

One method for synchronizing the primary and standby databases is to manually copy the primary database log files into the standby database log path to be used for local catchup. If you synchronize the primary and standby databases by manually copying the primary database logs into the standby database log path, you must copy the primary log files before you start the standby database for the following reasons:

1. When the end of the local log files is reached, the standby database will enter remote catchup pending state and will not try to access the local log files again until the standby database is restarted.
2. If the standby database enters remote catchup state, copying log files into its log path could interfere with the writing of local log files by the standby database.

Determining the HADR standby database state using the GET SNAPSHOT command

You can determine the state of a DB2 High Availability Disaster Recovery (HADR) standby database by issuing the GET SNAPSHOT command with the DATABASE ON option.

To determine the state of na HADR standby database in a primary-standby HADR database pair, you can issue the GET SNAPSHOT command from the primary database or the standby database.

- If you issue the GET SNAPSHOT command from the standby database, the state of the standby database is returned in the `State` field of the output.
- If you issue the GET SNAPSHOT command from a primary database that is connected to the standby database, the state of the standby database is returned in the `State` field of the output.
- If you issue the GET SNAPSHOT command from a primary database that is not connected to the standby database, `disconnected` is returned in the `State` field of the output.

For example, if you have standby database MUSIC, you can issue the following command to see its state:

```
get snapshot for database on music
```

The following output shows the HADR status section returned by the GET SNAPSHOT command:

```
HADR status

  Role                 = Primary
  State                = Peer
  Synchronization mode = Sync
```

```
Connection status      = Connected, 11-03-2002 12:23:09.35092
Heartbeat missed       = 0
Local host             = host1.ibm.com
Local service          = hadr_service
Remote host            = host2.ibm.com
Remote service         = hadr_service
Remote instance        = dbinst2
timeout(seconds)       = 120
Primary log position(file, page, LSN) = S0001234.LOG, 12, 0000000000BB800C
Standby log position(file, page, LSN) = S0001234.LOG, 12, 0000000000BB800C
Log gap running average(bytes) = 8723
```

While reviewing the output of the GET SNAPSHOT command, you might notice a log gap. A log gap can happen because when a log file is truncated, either as the result of an explicit log truncation, or as a result of stopping and restarting the primary database, the primary moves to the beginning of the next log file. The standby, however, stays at the end of the last log file. As soon as the primary writes any log, the log will be replicated and the standby will update its log position.

DB2 High Availability Disaster Recovery (HADR) management

DB2 High Availability Disaster Recovery (HADR) management involves configuring and maintaining the status of your HADR system.

Managing HADR includes such tasks as:
- “Initializing high availability disaster recovery (HADR)” on page 31
- “Stopping DB2 High Availability Disaster Recovery (HADR)” on page 145
- “Switching database roles in high availability disaster recovery (HADR)” on page 168
- “Performing an HADR failover operation” on page 165
- “Monitoring high availability disaster recovery (HADR)” on page 161
- Checking or altering database configuration parameters related to HADR.
- Cataloging an HADR database (if required).

You can manage HADR using the following methods:
- Command line processor
- Control Center GUI tools
- DB2 administrative API

DB2 High Availability Disaster Recovery (HADR) commands

The DB2 High Availability Disaster Recovery (HADR) feature provides complex logging, failover, and recovery functionality for DB2 high availability database solutions. Despite the complexity of the functionality HADR provides, there are only a few actions you need to directly command HADR to perform: starting HADR; stopping HADR; and causing the standby database to take over as the primary database.

There are three high availability disaster recover (HADR) commands used to manage HADR:
- Start HADR
- Stop HADR
- Takeover HADR

To invoke these commands, use the command line processor or the administrative API. You can also invoke these commands using the GUIs available from the Manage High Availability Disaster Recovery window in the Control Center. To open the Manage High Availability Disaster Recovery window from the Control Center, right-click a database and click High Availability Disaster Recovery—–>Manage.

Issuing the START HADR command with either the AS PRIMARY or AS STANDBY option changes the database role to the one specified if the database is not already in that role. This command also activates the database, if it is not already activated.

The STOP HADR command changes an HADR database (either primary or standby) into a standard database. Any database configuration parameters related to HADR remain unchanged so that the database can easily be reactivated as an HADR database.

The TAKEOVER HADR command, which you can issue on the standby database only, changes the standby database to a primary database. When you do not specify the BY FORCE option, the primary and standby databases switch roles. When you do specify the BY FORCE option, the standby database unilaterally switches to become the primary database. In this case, the standby database attempts to stop transaction processing on the old primary database. However, there is no guarantee that transaction processing will stop. Use the BY FORCE option to force a takeover operation for failover conditions only. To whatever extent possible, ensure that the current primary has definitely failed, or shut it down yourself, prior to issuing the TAKEOVER HADR command with the BY FORCE option.

HADR database role switching

A database can be switched between primary and standard roles dynamically and repeatedly. When the database is either online or offline, you can issue both the START HADR command with the AS PRIMARY option and the STOP HADR command.

You can switch a database between standby and standard roles statically. You can do so repeatedly only if the database remains in rollforward pending state. You can issue the START HADR command with the AS STANDBY option to change a standard database to standby while the database is offline and in rollforward pending state. Use the STOP HADR command to change a standby database to a standard database while the database is offline. The database remains in rollforward pending state after you issue the STOP HADR command. Issuing a subsequent START HADR command with the AS STANDBY option returns the database to standby. If you issue the ROLLFORWARD DATABASE command with the STOP option after stopping HADR on a standby database, you cannot bring it back to standby. Because the database is out of rollforward pending state, you can use it as a standard database. This is referred to as taking a snapshot of the standby database. After changing an existing standby database into a standard database, consider creating a new standby database for high availability purposes.

To switch the role of the primary and standby databases, perform a takeover operation without using the BY FORCE option.

To change the standby to primary unilaterally (without changing the primary to standby), use forced takeover. Subsequently, you might be able to reintegrate the old primary as a new standby.

HADR role is persistent. Once an HADR role is established, it remains with the database, even through repeated stopping and restarting of the DB2 instance or deactivation and activation of the DB2 database.

Starting the standby is asynchronous

When you issue the START HADR command with the AS STANDBY option, the command returns as soon as the relevant engine dispatchable units (EDUs) are successfully started. The command does not wait for the standby to connect to the primary database. In contrast, the primary database is not considered started until it connects to a standby database (with the exception of when the START HADR command is issued on the primary with the BY FORCE option). If the standby database encounters an error, such as the connection being rejected by the primary database, the START HADR command with the AS STANDBY option might have already returned successfully. As a result, there is no user prompt to which HADR can return an error indication. The HADR standby will write a message to the DB2 diagnostic log and shut itself down. You should monitor the status of the HADR standby to ensure that it successfully connects with the HADR primary.

Replay errors, which are errors that the standby encounters while replaying log records, can also bring down the standby database. These errors might occur, for example, when there is not enough memory to create a buffer pool, or if the path is not found while creating a table space. You should continuously monitor the status of the standby database.

Do not run HADR commands from a client using a database alias enabled for client reroute

When automatic client reroute is set up, the database server has a predefined alternate server so that client applications can switch between working with either the original database server or the alternative server with only minimal interruption of the work. In such an environment, when a client connects to the database via TCP, the actual connection can go to either the original database or to the alternate database. HADR commands are implemented to identify the target database through regular client connection logic. Consequently, if the target database has an alternative database defined, it is difficult to determine the database on which the command is actually operating. Although an SQL client does not need to know which database it is connecting to, HADR commands must be applied on a specific database. To accommodate this limitation, HADR commands should be issued locally on the server machine so that client reroute is bypassed (client reroute affects only TCP/IP connections).

HADR commands must be run on a server with a valid license

The START HADR, STOP HADR, and TAKEOVER HADR commands require that a valid HADR license has been installed on the server where the command is executed. If the license is not present, these commands will fail and return a command-specific error code (SQL1767N, SQL1769N, or SQL1770N, respectively) along with a reason code of 98. To correct the problem, either install a valid HADR license using db2licm, or install a version of the server that contains a valid HADR license as part of its distribution.

Chapter 6. Detecting and responding to system outages in a high availability solution

Implementing a high availability solution does not prevent hardware or software failures. However, having redundant systems and a failover mechanism enables your solution to detect and respond to failures, and reroute workload so that user applications are still able to do work.

When a failure occurs, your database solution must do the following:

1. Detect the failure.

   Failover software can use heartbeat monitoring to confirm the availability of system components. A heartbeat monitor listens for regular communication from all the components of the system. If the heartbeat monitor stops hearing from a component, the heartbeat monitor signals to the system that the component has failed.
2. Respond to the failure: failover.
   a. Identify, bring online, and initialize a secondary component to take over operations for the failed component.
   b. Reroute workload to the secondary component.
   c. Remove the failed component from the system.
3. Recover from the failure.

   If a primary database server fails, the first priority is to redirect clients to an alternate server or to failover to a standby database so that client applications can do their work with as little interruption as possible. Once that failover succeeds, you must repair whatever went wrong on the failed database server so that is can be reintegrate it back into the solution. Repairing the failed database server may just mean restarting it.
4. Return to normal operations.

   Once the failed database system is repaired, you must integrate it back into the database solution. You could reintegrate a failed primary database as the standby database for the database that took over as the primary database when the failure occurred. You could also force the repaired database server to take over as the primary database server again.

DB2 database can perform some of these steps for you. For example:

- The DB2 High Availability Disaster Recovery (HADR) heartbeat monitor element, hadr_heartbeat, can detect when a primary database has failed.
- DB2 client reroute can transfer workload from a failed database server to a secondary one.
- The DB2 fault monitor can restart a database instance that terminates unexpectedly.

Administration notification log

The administration notification log (*instance_name*.nfy) is the repository from which information about numerous database administration and maintenance activities can be obtained. A database administrator can use this information to diagnose problems, tune the database, or simply monitor the database.

The DB2 database manager writes the following kinds of information to the administration notification log on UNIX and Linux operating system platforms (on Windows operating system platforms, the event log is used to record administration notification events):

- Status of DB2 utilities such REORG and BACKUP
- Client application errors
- Service class changes
- Licensing activity
- File paths
- Storage problems
- Monitoring activities
- Indexing activities
- Table space problems

Administration notification log messages are also logged to the db2diag log files using a standardized message format.

Notification messages provide additional information to supplement the SQLCODE that is provided.

The administration notification log file can exist in two different forms:

Single administration notification log file
One active administration notification log file, named *instance_name*.nfy, that grows in size indefinitely. This is the default form and it exists whenever the **diagsize** database manager configuration parameter has the value of 0 (the default value for this parameter is 0).

Rotating administration notification log files
A single active log file (named *instance_name*.*N*.nfy, where *N* is the file name index that is a continuously growing number starting from 0), although a series of administration notification log files can be found in the location defined by the **diagpath** configuration parameter, each growing until reaching a limited size, at which time the log file is closed and a new one is created and opened for logging with an incremented file name index (*instance_name*.*N+1*.nfy). It exists whenever the **diagsize** database manager configuration parameter has a nonzero value.

Note: Neither single nor rotating administration notification log files are available on the Windows operating system platform.

You can choose which of these two forms exist on your system by appropriately setting the **diagsize** database manager configuration parameter.

Configurations

The administration notification log files can be configured in size, location, and the types of events and level of detail recorded, by setting the following database manager configuration parameters:

diagsize
The value of **diagsize** decides what form of administration notification log file will be adopted. If the value is 0, a single administration notification log file will be adopted. If the value is not 0, rotating administration notification log files will be adopted, and this nonzero value also specifies

the total size of all rotating diagnostic log files and all rotating administration notification log files. The instance must be restarted for the new value of the **diagsize** parameter to take effect. See the ″diagsize - Diagnostic log file size configuration parameter″ topic for complete details.

diagpath
Diagnostic information can be specified to be written to administration notification log files in the location defined by the **diagpath** configuration parameter. See the ″diagpath - Diagnostic data directory path configuration parameter″ topic for complete details.

notifylevel
The types of events and the level of detail written to the administration notification log files can be specified with the **notifylevel** configuration parameter. See the ″notifylevel - Notify level configuration parameter″ topic for complete details.

Detecting an unplanned outage

Before you can respond to the failure of a component, you must detect that the component failed. DB2 Data Server has several tools for monitoring the health of a database, or otherwise detecting that a database has failed. You can configure these tools to notify you or take predefined actions when they detect a failure.

You can use the following tools to detect when a failure has occurred in some part of your DB2 database solution:

DB2 fault monitor facility

The DB2 fault monitor facility keeps DB2 database instances up and running. When the DB2 database instance to which a DB2 fault monitor is attached terminates unexpectedly, the DB2 fault monitor restarts the instance. If your database solution is implemented in a cluster, you should configure the cluster managing software to restart failed database instances instead of the DB2 fault monitor.

Heartbeat monitoring in clustered environments

Cluster managing software uses heartbeat messages between the nodes of a cluster to monitor the health of the nodes. The cluster manager detects that a node has failed when the node stops responding or sending any messages.

Monitoring DB2 High Availability Disaster Recovery (HADR) databases

The HADR feature has its own heartbeat monitor. The primary database and the standby database each expect heartbeat messages from the other at regular intervals.

Monitoring high availability disaster recovery (HADR)

You can use the following methods to monitor the status of your HADR databases.

db2pd utility
This utility retrieves information from the DB2 memory sets. For example, to view information about high availability disaster recovery for database MYDB, issue the following:

```
db2pd -db mydb -hadr
```

GET SNAPSHOT FOR DATABASE command
This command collects status information and formats the output. The

information returned represents a snapshot of the database manager operational status at the time the command was issued. HADR information appears in the output under the heading *HADR status*.

db2GetSnapshot API
This API collects database manager monitor information and returns it to a user-allocated data buffer. The information returned represents a snapshot of the database manager operational status at the time the API was called.

HADR configuration parameters are not dynamic

If you change a parameter while the HADR database is online, the changes are visible when you issue a db2 get db cfg for the database. However, the changes are not effective until you stop and restart the database. To retrieve the parameters that are currently effective, use the GET SNAPSHOT command, the db2pd tool, or the snapshot monitor API.

HADR database roles

The current role of a database is indicated by the database configuration parameter *hadr_db_role*. Valid values for this configuration parameter are PRIMARY, STANDBY, or STANDARD (the latter indicates the database is not an HADR database).

Status of the standby database

When a database is in the standby role, it is also in rollforward pending state. Consequently, the standby database configuration will indicate:

```
Rollforward pending
           = DATABASE
Restore pending                                       = YES
```

Responding to an unplanned outage

If your database management software or cluster management software detects that a database server has failed, your database solution must respond to that failure as quickly and as smoothly as possible. Your database solution must attempt to shield user applications from the failure by rerouting workload, if possible, and failover to a secondary or standby database, if one is available.

If your database or cluster management software detects that a database server has failed, you or your database or cluster management software must do the following:

1. Identify, bring online, and initialize a secondary database server to take over operations for the failed database server.

   If you are using DB2 High Availability Disaster Recover (HADR) to manage primary and standby database servers, HADR will manage keeping the standby database synchronized with the primary database; and HADR will manage the takeover of the primary database by the standby database.
2. Reroute user application workload to the secondary database server.

   DB2 client reroute can automatically reroute client application away from a failed database server to a secondary database server previously identified and configured for this purpose.
3. Remove the failed database server from the system to repair it.

Once the user applications have been rerouted to a secondary or standby database server, the failed database server can not handle any client application requests until it has been restarted or otherwise repaired. For example, if the cause of the failure on the primary database was that a database instance terminated unexpectedly, the DB2 fault monitor facility will automatically restart it.

Automatic client reroute examples

DB2 Data Server client reroute can automatically reroute client application away from a failed database server to a secondary database server previously identified and configured for this purpose. You can easily create client applications that test and demonstrate this DB2 Data Server functionality.

Here is an automatic client reroute example for a client application (shown using pseudo-code only):

```
int checkpoint = 0;

check_sqlca(unsigned char *str, struct sqlca *sqlca)
{
   if (sqlca->sqlcode == -30081)
   {
      // as communication is lost, terminate the application right away
      exit(1);
   }
   else

      // print out the error
      printf(...);

 if (sqlca->sqlcode == -30108)
 {
    // connection is re-established, re-execute the failed transaction
         if (checkpoint == 0)
         {
             goto checkpt0;
         }
    else if (checkpoint == 1)
         {
            goto checkpt1;
         }
         else if (checkpoint == 2)
         {
             goto checkpt2;
         }
         ....
         exit;
 }
 }
}

main()
{
   connect to mydb;
   check_sqlca("connect failed", &sqlca);

checkpt0:
   EXEC SQL set current schema XXX;
   check_sqlca("set current schema XXX failed", &sqlca);

   EXEC SQL create table t1...;
   check_sqlca("create table t1 failed", &sqlca);

   EXEC SQL commit;
```

```
      check_sqlca("commit failed", &sqlca);

      if (sqlca.sqlcode == 0)
      {
         checkpoint = 1;
      }

   checkpt1:
      EXEC SQL set current schema YYY;
      check_sqlca("set current schema YYY failed", &sqlca);

      EXEC SQL create table t2...;
      check_sqlca("create table t2 failed", &sqlca);

      EXEC SQL commit;
      check_sqlca("commit failed", &sqlca);

      if (sqlca.sqlcode == 0)
      {
         checkpoint = 2;
      }
   ...
   }
```

At the client machine, the database called "mydb" is cataloged which references a node "hornet" where "hornet" is also cataloged in the node directory (hostname "hornet" with port number 456).

Example 1 (involving a non-HADR database)

At the server "hornet" (hostname equals hornet with a port number), a database "mydb" is created. Furthermore, the database "mydb" is also created at the alternate server (hostname "montero" with port number 456). You will also need to update the alternate server for database "mydb" at server "hornet" as follows:

```
db2 update alternate server for database mydb using hostname montero port 456
```

In the sample application above, and without having the automatic client reroute feature set up, if there is a communication error in the `create table t1` statement, the application will be terminated. With the automatic client reroute feature set up, the DB2 database manager will try to establish the connection to host "hornet" (with port 456) again. If it is still not working, the DB2 database manager will try the alternate server location (host "montero" with port 456). Assuming there is no communication error on the connection to the alternate server location, the application can then continue to run subsequent statements (and to re-run the failed transaction).

Example 2 (involving an HADR database)

At the server "hornet" (hostname equals hornet with a port number), primary database "mydb" is created. A standby database is also created at host "montero" with port 456. Information on how to setup HADR for both a primary and standby database is found in *Data Recovery and High Availability Guide and Reference*. You will also need to update the alternate server for database "mydb" as follows:

```
db2 update alternate server for database mydb using hostname montero port 456
```

In the sample application, above, and without having the automatic client reroute feature set up, if there is a communication error in the `create table t1` statement, the application will be terminated. With the automatic client reroute feature set up, the DB2 database system will try to establish the connection to host "hornet" (with port 456) again. If it is still not working, the DB2 database system will try the alternate server location (host “montero” with port 456). Assuming there is no communication error on the connection to the alternate server location, the application can then continue to run subsequent statements (and to re-run the failed transaction).

Example 3 (involving SSL)

You can also use client reroute while you are using SSL for your connections too. The setup is the similar to that shown for **Example2**.

At the client machine, a database alias “mydb_ssl” for the database “mydb” is cataloged that references a node, “hornet_ssl”. “hornet_ssl” is cataloged in the node directory (hostname is “hornet”, SSL port number is 45678, and the security parameter is set to SSL).

A database alias is also cataloged at the alternate server (hostname is “montero”, SSL port number is 45678, and the security parameter is set to SSL). You also need to update the alternate server for alias “mydb_ssl” at server “hornet” as shown:

```
db2 update alternate server for database mydb_ssl using hostname montero port 45678
```

In the sample application, above, change the connect statement to `connect to mydb_ssl`. Without having the automatic client reroute feature set up, if there is a communication error in the `create table t1` statement, the application will be terminated. With the automatic client reroute feature set up, the DB2 database manager tries to establish the connection to host “hornet” (with port 45678) using SSL again. If it is still does not work, the DB2 database manager tries the alternate server location (host “montero” at port 45678) using SSL. Assuming there is no communication error on the connection to the alternate server location, the application can then continue to run subsequent statements (and to re-run the failed transaction).

Performing an HADR failover operation

When you want the current standby database to become the new primary database because the current primary database is not available, you can perform a failover.

Warning:

This procedure might cause a loss of data. Review the following information before performing this emergency procedure:

- Ensure that the primary database is no longer processing database transactions. If the primary database is still running, but cannot communicate with the standby database, executing a forced takeover operation (issuing the TAKEOVER HADR command with the BY FORCE option) could result in two primary databases. When there are two primary databases, each database will have different data, and the two databases can no longer be automatically synchronized.
  - Deactivate the primary database or stop its instance, if possible. (This might not be possible if the primary system has hung, crashed, or is otherwise inaccessible.) After a takeover operation is performed, if the failed database is later restarted, it will not automatically assume the role of primary database.
- The likelihood and extent of transaction loss depends on your specific configuration and circumstances:

  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is synchronous (SYNC), the standby database will not lose transactions that were reported committed to an application before the primary database failed.
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is near synchronous (NEARSYNC), the standby database can only lose transactions committed by the primary database if both the primary and the standby databases fail at the same time.
  - If the primary database fails while in peer state or disconnected peer state and the synchronization mode is asynchronous (ASYNC), the standby database can lose transactions committed by the primary database if the standby database did not receive all of the log records for the transactions before the takeover operation was performed. The standby database can also lose transactions committed by the primary database if both the primary and the standby databases fail at the same time.
  - If the primary database fails while in remote catchup pending state, transactions that have not been received and processed by the standby database will be lost.

    Note: Any log gap shown in the database snapshot will represent the gap at the last time the primary and standby databases were communicating with each other; the primary database might have processed a very large number of transactions since that time.
- Ensure that any application that connects to the new primary (or that is rerouted to the new primary by client reroute), is prepared to handle the following:
  - There is data loss during failover. The new primary does not have all of the transactions committed on the old primary. This can happen even when the HADR_SYNCMODE configuration parameter is set to SYNC. Because an HADR standby applies logs sequentially, you can assume that if a transaction in an SQL session is committed on the new primary, all previous transactions in the same session have also been committed on the new primary. The commit sequence of transactions across multiple sessions can be determined only with detailed analysis of the log stream.
  - It is possible that a transaction can be issued to the original primary, committed on the original primary and replicated to the new primary (original standby), but not be reported as committed because the original primary crashed before it could report to the client that the transaction was committed. Any application you write should be able to handle that transactions issued to the original primary, but not reported as committed on the original primary, are committed on the new primary (original standby).
  - Some operations are not replicated, such as changes to database configuration and to external UDF objects.
- The TAKEOVER HADR command can only be issued on the standby database.
- HADR does not interface with the DB2 fault monitor (db2fm) which can be used to automatically restart a failed database. If the fault monitor is enabled, you should be aware of possible fault monitor action on a presumably failed primary database.
- A takeover operation can only take place if the primary and standby databases are in peer state or the standby database is in remote catchup pending state. If the standby database is in any other state, an error will be returned.

  Note: You can make a standby database that is in local catchup state available for normal use by converting it to a standard database. To do this, shut the

database down by issuing the DEACTIVATE DATABASE command, and then issue the STOP HADR command. Once HADR has been stopped, you must complete a rollforward operation on the former standby database before it can be used. A database cannot rejoin an HADR pair after it has been converted from a standby database to a standard database. To restart HADR on the two servers, follow the procedure for initializing HADR.
If you have configured a peer window, shut down the primary before the window expires to avoid potential transaction loss in any related failover.

In a failover scenario, a takeover operation can be performed through the command line processor (CLP), the Manage High Availability Disaster Recovery window in the Control Center, or the db2HADRTakeover application programming interface (API).

The following procedure shows you how to initiate a failover on the primary or standby database using the CLP:

1. Completely disable the failed primary database. When a database encounters internal errors, normal shutdown commands might not completely shut it down. You might need to use operating system commands to remove resources such as processes, shared memory, or network connections.
2. Issue the TAKEOVER HADR command with the BY FORCE option on the standby database. In the following example the failover takes place on database LEAFS:

   ```
   TAKEOVER HADR ON DB LEAFS BY FORCE
   ```

   The BY FORCE option is required because the primary is expected to be offline.

   If the primary database is not completely disabled, the standby database will still have a connection to the primary and will send a message to the primary database asking it to shutdown. The standby database will still switch to the role of primary database whether or not it receives confirmation from that the primary database has been shutdown.

To open the Takeover HADR window:

1. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.
2. Click Takeover HADR. The Takeover HADR window opens.
3. Select that you want to execute a failover operation.
4. If both databases in the HADR pair have been started as standby databases, select one of the databases to take over as the primary database.
5. Click OK. The window closes. A progress indicator might open to indicate when the command is running. When it completes, you will get a notification indicating whether or not it is successful.
6. Refresh the Manage High Availability Disaster Recovery window to ensure that the standby database has taken over as the new primary.
7. If you are not using the automatic client reroute feature, redirect client applications to the new primary database.

Detailed information is provided through the online help facility within the Control Center.

Switching database roles in high availability disaster recovery (HADR)

During high availability disaster recovery (HADR), use the TAKEOVER HADR command to switch the roles of the primary and standby databases.

- The TAKEOVER HADR command can only be issued on the standby database. If the primary database is not connected to the standby database when the command is issued, the takeover operation will fail.
- The TAKEOVER HADR command can only be used to switch the roles of the primary and standby databases if the databases are in peer state. If the standby database is in any other state, an error message will be returned.

You can switch the HADR database roles using the command line processor (CLP), the Manage High Availability Disaster Recovery (HADR) window in the Control Center, or the db2HADRTakeover application programming interface (API).

To use the CLP to initiate a takeover operation on the standby database, issue the TAKEOVER HADR command without the BY FORCE option on the standby database.

In the following example, the takeover operation takes place on the standby database LEAFS:

```
TAKEOVER HADR ON DB LEAFS
```

A log full error is slightly more likely to occur immediately following a takeover operation. To limit the possibility of such an error, an asynchronous buffer pool flush is automatically started at the end of each takeover. The likelihood of a log full error decreases as the asynchronous buffer pool flush progresses. Additionally, if your configuration provides a sufficient amount of active log space, a log full error is even more unlikely. If a log full error does occur, the current transaction is aborted and rolled back.

Note: Issuing the TAKEOVER HADR command without the BY FORCE option will cause any applications currently connected to the HADR primary database to be forced off. This action is designed to work in coordination with automatic client reroute to assist in rerouting clients to the new HADR primary database after a role switch. However, if the forcing off of applications from the primary would be disruptive in your environment, you might want to implement your own procedure to shut down such applications prior to performing a role switch, and then restart them with the new HADR primary database as their target after the role switch is completed.

To open the Takeover HADR window:

1. From the Control Center, expand the object tree until you find the database for which you want to manage HADR. Right-click the database and click High Availability Disaster Recovery→Manage in the pop-up menu. The Manage High Availability Disaster Recovery window opens.
2. Ensure that the databases are in peer state
3. Click Takeover HADR. The Takeover HADR window opens.
4. Select that you want to switch the database roles.
5. If both databases in the HADR pair have been started as standby databases, select one of the databases to take over as the primary database.

6. Click OK. The window closes. A progress indicator might open to indicate when the command is running. When it completes, you will get a notification indicating whether or not it is successful.
7. Refresh the Manage High Availability Disaster Recovery window to ensure that the databases have switched roles.
8. If you are not using the automatic client reroute feature, redirect client applications to the new primary database.

   Additional information is provided through the contextual help facility within the Control Center.

Reintegrating a database after a takeover operation

If you executed a takeover operation in a high availability disaster recovery (HADR) environment because the primary database failed, you can bring the failed database back online and use it as a standby database or return it to its status as primary database.

To reintegrate the failed primary database into the HADR pair as the new standby database:

1. Repair the system where the original primary database resided. This could involve repairing failed hardware or rebooting the crashed operating system.
2. Restart the failed primary database as a standby database. In the following example, database LEAFS is started as a standby database:

   ```
   START HADR ON DB LEAFS AS STANDBY
   ```

   Note: Reintegration will fail if the two copies of the database have incompatible log streams. In particular, HADR requires that the original primary database did not apply any logged operation that was never reflected on the original standby database before it took over as the new primary database. If this did occur, you can restart the original primary database as a standby database by restoring a backup image of the new primary database or by initializing a split mirror.

   Successful return of this command does not indicate that reintegration has succeeded; it means only that the database has been started. Reintegration is still in progress. If reintegration subsequently fails, the database will shut itself down. You should monitor standby states using the GET SNAPSHOT FOR DATABASE command or the db2pd tool to make sure that the standby database stays online and proceeds with the normal state transition. If necessary, you can check the administration notification log file and the db2diag log file to find out the status of the database.

After the original primary database has rejoined the HADR pair as the standby database, you can choose to perform a failback operation to switch the roles of the databases to enable the original primary database to be once again the primary database. To perform this failback operation, issue the following command on the standby database:

```
TAKEOVER HADR ON DB LEAFS
```

Note:

1. If the HADR databases are not in peer state or the pair is not connected, this command will fail.
2. Open sessions on the primary database are forced closed and inflight transactions are rolled back.

3. When switching the roles of the primary and standby databases, the BY FORCE option of the TAKEOVER HADR command cannot be specified.

Part 2. Data recovery

Recovery is the rebuilding of a database or table space after a problem such as media or storage failure, power interruption, or application failure. If you have backed up your database, or individual table spaces, you can rebuild them should they become damaged or corrupted in some way.

There are three types of recovery:

- Crash recovery protects a database from being left in an inconsistent, or unusable, state when transactions (also called units of work) are interrupted unexpectedly.
- Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation.
- Rollforward recovery can be used to reapply changes that were made by transactions that were committed after a backup was made.

The DB2 database manager starts crash recovery automatically to attempt to recover a database after a power interruption. You can use version recovery or rollforward recovery to recover a damaged database.

Chapter 7. Developing a backup and recovery strategy

A database can become unusable because of hardware or software failure, or both. You might, at one time or another, encounter storage problems, power interruptions, or application failures, and each failure scenario requires a different recovery action. Protect your data against the possibility of loss by having a well rehearsed recovery strategy in place. Some of the questions that you should answer when developing your recovery strategy are:

- Will the database be recoverable?
- How much time can be spent recovering the database?
- How much time will pass between backup operations?
- How much storage space can be allocated for backup copies and archived logs?
- Will table space level backups be sufficient, or will full database backups be necessary?
- Should I configure a standby system, either manually or through high availability disaster recovery (HADR)?

A database recovery strategy should ensure that all information is available when it is required for database recovery. It should include a regular schedule for taking database backups and, in the case of partitioned database environments, include backups when the system is scaled (when database partition servers or nodes are added or dropped). Your overall strategy should also include procedures for recovering command scripts, applications, user-defined functions (UDFs), stored procedure code in operating system libraries, and load copies.

Different recovery methods are discussed in the sections that follow, and you will discover which recovery method is best suited to your business environment.

The concept of a database *backup* is the same as any other data backup: taking a copy of the data and then storing it on a different medium in case of failure or damage to the original. The simplest case of a backup involves shutting down the database to ensure that no further transactions occur, and then simply backing it up. You can then recreate the database if it becomes damaged or corrupted in some way.

The recreation of the database is called *recovery*. *Version recovery* is the restoration of a previous version of the database, using an image that was created during a backup operation. *Rollforward recovery* is the reapplication of transactions recorded in the database log files after a database or a table space backup image has been restored.

Crash recovery is the automatic recovery of the database if a failure occurs before all of the changes that are part of one or more units of work (transactions) are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

Recovery log files and the recovery history file are created automatically when a database is created (Figure 10 on page 174). These log files are important if you need to recover data that is lost or damaged.

Each database includes *recovery logs*, which are used to recover from application or system errors. In combination with the database backups, they are used to recover the consistency of the database right up to the point in time when the error occurred.

The *recovery history file* contains a summary of the backup information that can be used to determine recovery options, if all or part of the database must be recovered to a given point in time. It is used to track recovery-related events such as backup and restore operations, among others. This file is located in the database directory.

The *table space change history file*, which is also located in the database directory, contains information that can be used to determine which log files are required for the recovery of a particular table space.

You cannot directly modify the recovery history file or the table space change history file; however, you can delete entries from the files using the PRUNE HISTORY command. You can also use the *rec_his_retentn* database configuration parameter to specify the number of days that these history files will be retained.

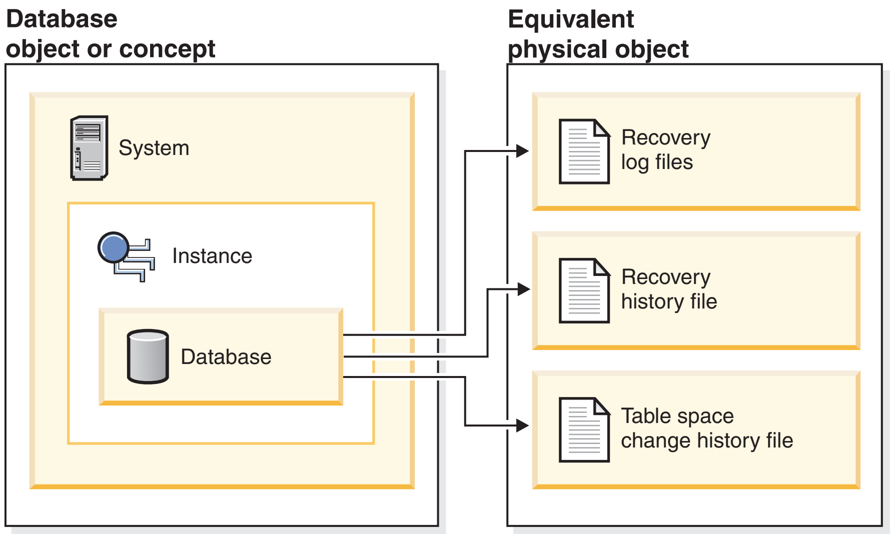

Database object or concept
Equivalent physical object
System
Instance
Database
Recovery log files
Recovery history file
Table space change history file

Figure 10. Database recovery files

Data that is easily recreated can be stored in a non-recoverable database. This includes data from an outside source that is used for read-only applications, and tables that are not often updated, for which the small amount of logging does not justify the added complexity of managing log files and rolling forward after a restore operation. If both the *logarchmeth1* and *logarchmeth2* database configuration parameters are set to “OFF” then the database is *Non-recoverable*. This means that the only logs that are kept are those required for crash recovery. These logs are known as *active logs*, and they contain current transaction data. Version recovery using *offline* backups is the primary means of recovery for a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken and rollforward recovery is not supported.

Data that *cannot* be easily recreated should be stored in a recoverable database. This includes data whose source is destroyed after the data is loaded, data that is manually entered into tables, and data that is modified by application programs or users after it is loaded into the database. *Recoverable databases* have the *logarchmeth1* or *logarchmeth2* database configuration parameters set to a value other than "OFF". Active logs are still available for crash recovery, but you also have the *archived logs*, which contain committed transaction data. Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken. However, with rollforward recovery, you can roll the database *forward* (that is, past the time when the backup image was taken) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

Recoverable database backup operations can be performed either offline or *online* (online meaning that other applications can connect to the database during the backup operation). Online table space restore and rollforward operations are supported only if the database is recoverable. If the database is non-recoverable, database restore and rollforward operations must be performed offline. During an online backup operation, rollforward recovery ensures that *all* table changes are captured and reapplied if that backup is restored.

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or rollforward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

Automated backup operations

Since it can be time-consuming to determine whether and when to run maintenance activities such as backup operations, you can use the Configure Automatic Maintenance wizard to do this for you. With automatic maintenance, you specify your maintenance objectives, including when automatic maintenance can run. DB2 then uses these objectives to determine if the maintenance activities need to be done and then runs only the required maintenance activities during the next available maintenance window (a user-defined time period for the running of automatic maintenance activities).

Note: You can still perform manual backup operations when automatic maintenance is configured. DB2 will only perform automatic backup operations if they are required.

Deciding how often to back up

Your recovery plan should allow for regularly scheduled backup operations, because backing up a database requires time and system resources. Your plan might include a combination of full database backups and incremental backup operations.

You should take full database backups regularly, even if you archive the logs (which allows for rollforward recovery). To recover a database, you can use either a full database backup image that contains all of the table space backup images, or you can rebuild the database using selected table space images. Table space backup images are also useful for recovering from an isolated disk failure or an application error. In partitioned database environments, you only need to restore the table spaces that reside on database partitions that have failed. You do not need to restore all of the table spaces or all of the database partitions.

Although full database backups are no longer required for database recovery now that you can rebuild a database from table space images, it is still good practice to occasionally take a full backup of your database.

You should also consider not overwriting backup images and logs, saving at least two full database backup images and their associated logs as an extra precaution.

If the amount of time needed to apply archived logs when recovering and rolling a very active database forward is a major concern, consider the cost of backing up the database more frequently. This reduces the number of archived logs you need to apply when rolling forward.

You can initiate a backup operation while the database is either *online* or *offline*. If it is online, other applications or processes can connect to the database, as well as read and modify data while the backup operation is running. If the backup operation is running offline, other applications *cannot* connect to the database.

To reduce the amount of time that the database is not available, consider using online backup operations. Online backup operations are supported only if rollforward recovery is enabled. If rollforward recovery is enabled and you have a complete set of recovery logs, you can restore the database, should the need arise. You can only use an online backup image for recovery if you have the logs that span the time during which the backup operation was running.

Offline backup operations are faster than online backup operations, since there is no contention for the data files.

The backup utility lets you back up selected table spaces. If you use DMS table spaces, you can store different types of data in their own table spaces to reduce the time required for backup operations. You can keep table data in one table space, long field and LOB data in another table space, and indexes in yet another table space. If you do this and a disk failure occurs, it is likely to affect only one of the table spaces. Restoring or rolling forward one of these table spaces will take less time than it would have taken to restore a single table space containing all of the data.

You can also save time by taking backups of different table spaces at different times, as long as the changes to them are not the same. So, if long field or LOB data is not changed as frequently as the other data, you can back up these table spaces less frequently. If long field and LOB data are not required for recovery, you can also consider not backing up the table space that contains that data. If the LOB data can be reproduced from a separate source, choose the NOT LOGGED option when creating or altering a table to include LOB columns.

Note: Consider the following if you keep your long field data, LOB data, and indexes in separate table spaces, but do not back them up together: If you back up a table space that does not contain all of the table data, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces that contain any type of data for a table must be rolled forward simultaneously to the same point in time.

If you reorganize a table, you should back up the affected table spaces after the operation completes. If you have to restore the table spaces, you will not have to roll forward through the data reorganization.

The time required to recover a database is made up of two parts: the time required to complete the restoration of the backup; and, if the database is enabled for forward recovery, the time required to apply the logs during the rollforward operation. When formulating a recovery plan, you should take these recovery costs and their impact on your business operations into account. Testing your overall recovery plan will assist you in determining whether the time required to recover the database is reasonable given your business requirements. Following each test, you might want to increase the frequency with which you take a backup. If rollforward recovery is part of your strategy, this will reduce the number of logs that are archived between backups and, as a result, reduce the time required to roll the database forward after a restore operation.

Storage considerations for recovery

When deciding which recovery method to use, consider the storage space required.

The version recovery method requires space to hold the backup copy of the database and the restored database. The rollforward recovery method requires space to hold the backup copy of the database or table spaces, the restored database, and the archived database logs.

If a table contains long field or large object (LOB) columns, you should consider placing this data into a separate table space. This will affect your storage space considerations, as well as affect your plan for recovery. With a separate table space for long field and LOB data, and knowing the time required to back up long field and LOB data, you might decide to use a recovery plan that only occasionally saves a backup of this table space. You can also choose, when creating or altering a table to include LOB columns, not to log changes to those columns. This will reduce the size of the required log space and the corresponding log archive space.

To prevent media failure from destroying a database and your ability to restore it, keep the database backup, the database logs, and the database itself on different devices. For this reason, it is highly recommended that you use the *newlogpath* configuration parameter to put database logs on a separate device once the database is created.

The database logs can use up a large amount of storage. If you plan to use the rollforward recovery method, you must decide how to manage the archived logs. Your choices are the following:

- Specify a log archiving method using the LOGARCHMETH1 or LOGARCHMETH2 configuration parameters.
- Manually copy the logs to a storage device or directory other than the database log path directory after they are no longer in the active set of logs.
- Use a user exit program to copy these logs to another storage device in your environment.

Keeping related data together

In the process of designing your database, you will develop an understanding of the relationships that exist between tables. These relationships can be expressed:

- At the application level, when transactions update more than one table
- At the database level, where referential integrity exists between tables, or where triggers on one table affect another table.

You should consider these relationships when developing a recovery plan. You will want to back up related sets of data together. Such sets can be established at either the table space or the database level. By keeping related sets of data together, you can recover to a point where all of the data is consistent. This is especially important if you want to be able to perform point-in-time rollforward recovery on table spaces.

Backup and restore operations between different operating systems and hardware platforms

DB2 database systems support some backup and restore operations between different operating systems and hardware platforms.

The supported platforms for DB2 backup and restore operations can be grouped into one of three families:

- Big-endian Linux and UNIX
- Little-endian Linux and UNIX
- Windows

A database backup from one platform family can only be restored on any system within the same platform family. For Windows operating systems, you can restore a database created on DB2 Universal Database (UDB) V8 on a DB2 Version 9 database system. For Linux and UNIX operating systems, as long as the endianness (big endian or little endian) of the backup and restore platforms is the same, you can restore backups that were produced on DB2 UDB V8 on DB2 Version 9.

The following table shows each of the Linux and UNIX platforms DB2 supports and indicates whether the platforms are big endian or little endian:

Table 9. Endianness of supported Linux and UNIX operating systems DB2 supports

Platform	Endianness
AIX	big endian
HP on IA64	big endian
Solaris x64	little endian
Solaris SPARC	big endian
Linux on zSeries®	big endian
Linux on pSeries®	big endian
Linux on IA-64	little endian
Linux on AMD64 and Intel® EM64T	little endian
32-bit Linux on x86	little endian

The target system must have the same (or later) version of the DB2 database product as the source system. You cannot restore a backup created on one version of the database product to a system running an earlier version of the database product. For example, you can restore a DB2 UDB V8 backup on a DB2 V9 database system, but you cannot restore a DB2 V9 backup on a DB2 UDB V8 database system.

Note: You can restore a database from a backup image taken on a 32–bit level into a 64–bit level, but not vice versa. The DB2 backup and restore utilities should be used to backup and restore your databases. Moving a fileset from one machine to another is not recommended as this may compromise the integrity of the database.

In situations where certain backup and restore combinations are not allowed, you can move tables between DB2 databases using other methods:

- db2move command
- Export utility followed by the import or the load utilities

Chapter 8. Recovery history file

A recovery history file is created with each database and is automatically updated whenever:

- A database or table spaces are backed up
- A database or table spaces are restored
- A database or table spaces are rolled forward
- A database is automatically rebuilt and more than one image is restored
- A table space is created
- A table space is altered
- A table space is quiesced
- A table space is renamed
- A table space is dropped
- A table is loaded
- A table is dropped (when dropped table recovery is enabled)
- A table is reorganized
- On-demand log archiving is invoked
- A new log file is written to (when using recoverable logging)
- A log file is archived (when using recoverable logging)
- A database is recovered

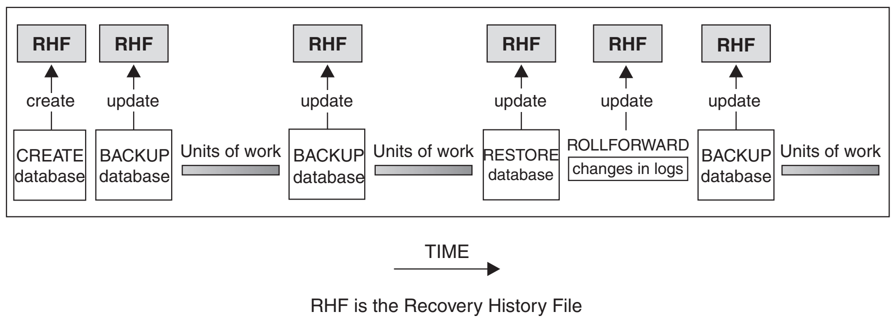


Figure 11. Creating and Updating the Recovery History File

You can use the summarized backup information in this file to recover all or part of a database to a given point in time. The information in the file includes:

- An identification (ID) field to uniquely identify each entry
- The part of the database that was copied and how
- The time the copy was made
- The location of the copy (stating both the device information and the logical way to access the copy)
- The last time a restore operation was done
- The time at which a table space was renamed, showing the previous and the current name of the table space
- The status of a backup operation: active, inactive, expired, or deleted

- The last log sequence number saved by the database backup or processed during a rollforward recovery operation.

To see the entries in the recovery history file, use the LIST HISTORY command.

Every backup operation (database, table space, or incremental) includes a copy of the recovery history file. The recovery history file is associated with the database. Dropping a database deletes the recovery history file. Restoring a database to a new location restores the recovery history file. Restoring does not overwrite the existing recovery history file unless the file that exists on disk has no entries. If that is the case, the database history will be restored from the backup image.

If the current database is unusable or not available, and the associated recovery history file is damaged or deleted, an option on the RESTORE command allows only the recovery history file to be restored. The recovery history file can then be reviewed to provide information on which backup to use to restore the database.

The size of the file is controlled by the *rec_his_retentn* configuration parameter that specifies a retention period (in days) for the entries in the file. Even if the number for this parameter is set to zero (0), the most recent full database backup (plus its restore set) is kept. (The only way to remove this copy is to use the PRUNE with FORCE option.) The retention period has a default value of 366 days. The period can be set to an indefinite number of days by using -1. In this case, explicit pruning of the file is required.

Recovery history file entry status

The database manager creates entries in the recovery history file for events such as a backup operation, a restore operation, table space creation, and others. Each entry in the recovery history file has an associated status: active, inactive, expired, deleted, or do_not_delete.

The database manager uses the status of a recovery history file entry to determine whether the physical files associated with that entry would be needed to recover the database. As part of automated pruning, the database manager updates the status of recovery history file entries.

Active database backup

An active database backup is one that can be restored and rolled forward using the current logs to recover the current state of the database.

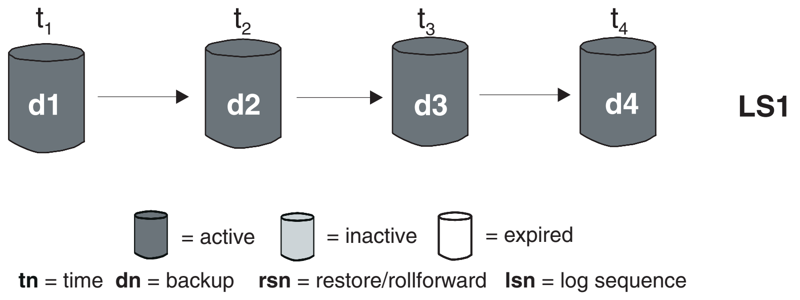
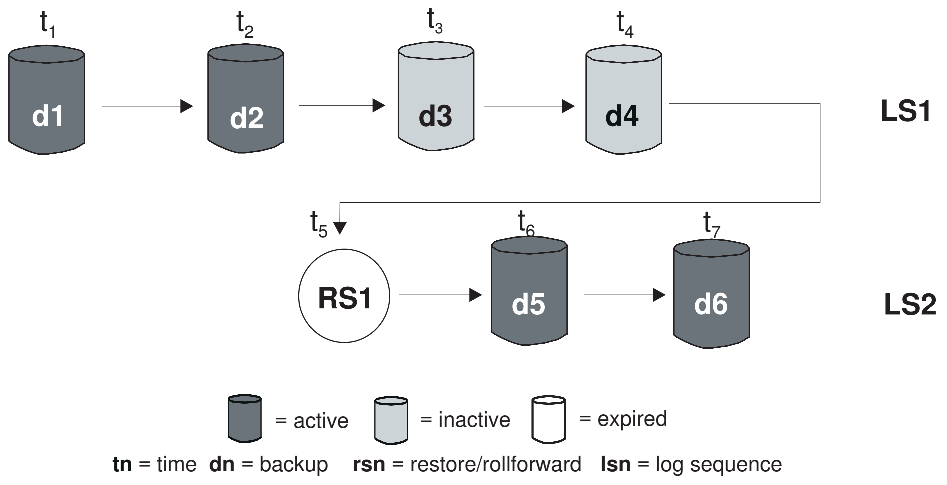

t1
t2
t3
t4
d1
d2
d3
d4
LS1
= active
= inactive
= expired
tn = time dn = backup rsn = restore/rollforward lsn = log sequence

Figure 12. Active Database Backups. The value of *num_db_backups* has been set to four.

Inactive database backup

An inactive database backup is one that, if restored, moves the database back to a previous state.

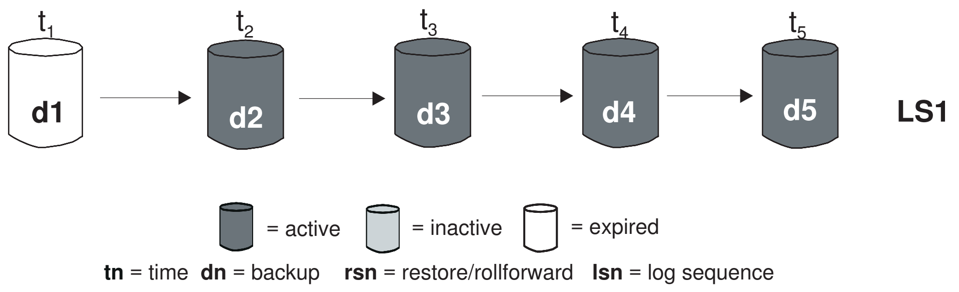


t1
t2
t3
t4
d1
d2
d3
d4
LS1
t5
t6
t7
RS1
d5
d6
LS2
= active
= inactive
= expired
tn = time dn = backup rsn = restore/rollforward lsn = log sequence

Figure 13. Inactive Database Backups

Expired database backups

An expired database backup image is one that is no longer needed, because more recent backup images are available.



t1
t2
t3
t4
t5
d1
d2
d3
d4
d5
LS1
= active
= inactive
= expired
tn = time dn = backup rsn = restore/rollforward lsn = log sequence

Figure 14. Expired Database Backups

Entries marked do_not_delete

You can remove (prune) recovery history file entries using the PRUNE HISTORY command or the db2Prune API. The database manager also prunes the recovery history file entries as part of automated pruning.

There are only three ways to prune entries marked do_not_delete:

- Invoke the PRUNE HISTORY command with the WITH FORCE option
- Call the ADMIN_CMD procedure with PRUNE HISTORY and WITH FORCE option
- Call the db2Prune API with the DB2_PRUNE_OPTION_FORCE option

Those entries that are marked do_not_delete will never be pruned from the recovery history file unless you perform one of these three actions.

The database manager does not set the status of recovery history file entries to do_not_delete. You can set the status of of a recovery history file entry to do_not_delete using the UPDATE HISTORY command.

Here are more examples of the status of different recovery history file entries:

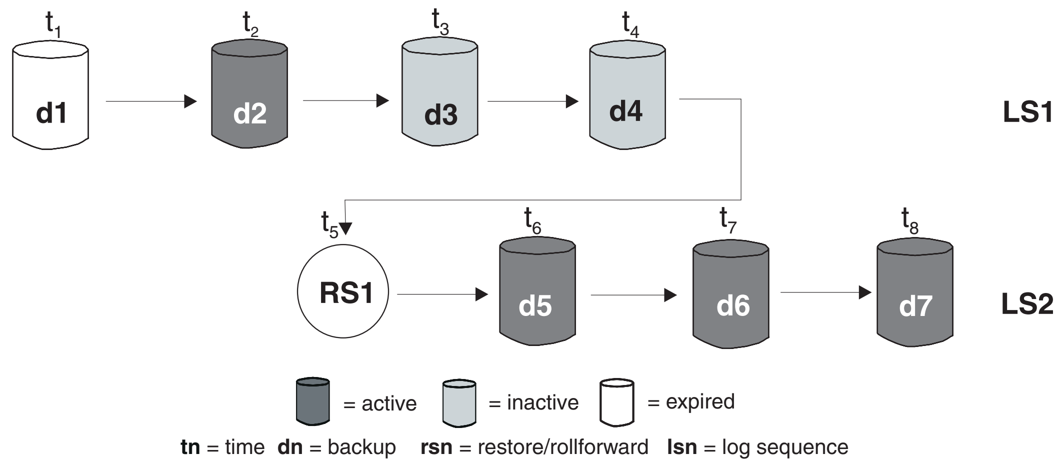

t1
t2
t3
t4
d1
d2
d3
d4
LS1
t5
t6
t7
t8
RS1
d5
d6
d7
LS2
= active
= inactive
= expired
tn = time dn = backup rsn = restore/rollforward lsn = log sequence

Figure 15. Mixed Active, Inactive, and Expired Database Backups

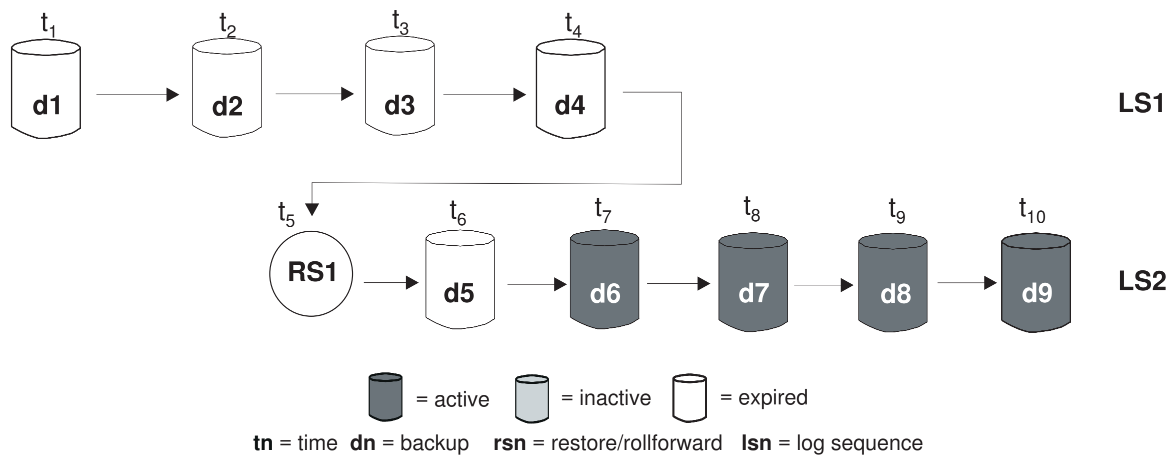


Figure 16. Expired Log Sequence

Viewing recovery history file entries using the DB_HISTORY administrative view

You can use the DB_HISTORY() administrative view to access the contents of the database history file. This method is an alternative to using the LIST HISTORY CLP command or the C history APIs.

A database connection is required to use this function.

Deletes and updates to the database history file can only be done through the PRUNE or UPDATE HISTORY commands.

This administrative view is not available for databases created using DB2 Universal Database Version 8.2 and earlier.

To access the history file:

1. Ensure you are connected to a database.
2. Use the DB_HISTORY() administrative view within an SQL SELECT statement to access the database history file for the database you are connected to, or on the database partition specified by the DB2NODE environment. For example, to see the contents of the history file use:
   ```
   SELECT * FROM TABLE(DB_HISTORY()) AS LIST_HISTORY
   ```

To hide the syntax of the administrative view, you can create a view as follows:

```
CREATE VIEW LIST_HISTORY AS
   SELECT * FROM TABLE(DB_HISTORY()) AS LIST_HISTORY
```

After creating this view, you can run queries against the view. For example:

```
SELECT * FROM LIST_HISTORY
```

or

```
SELECT dbpartitionnum FROM LIST_HISTORY
```

or

```
SELECT dbpartitionnum, start_time, seqnum, tabname, sqlstate
  FROM LIST_HISTORY
```

The Table 10 table lists the columns and the column data types returned by the LIST_HISTORY table function.

Table 10. Contents of the history table

Column name	Data type
dbpartitionnum smallint	smallint
EID	bigint
start_time	char(14)
seqnum	smallint
end_time	varchar(14)
firstlog	varchar(254)
lastlog	varchar(254)
backup_id	varchar(24)
tabschema	varchar(128)
tabname	varchar(128)
comment	varchar(254)
cmd_text	clob(2M)
num_tbsps	integer
tbspnames	clob(5M)
operation	char(1)
operationtype	char(1)

Table 10. Contents of the history table (continued)

Column name	Data type
objecttype	varchar(255)
location	char(1)
devicetype	char(1)
entry_status	varchar(8)
sqlcaid	varchar(8)
sqlcabc	integer
sqlcode	integer
sqlerrml	smallint
sqlerrmc	varchar(70)
sqlerrp	varchar(8)
sqlerrd1	integer
sqlerrd2	integer
sqlerrd3	integer
sqlerrd4	integer
sqlerrd5	integer
sqlwarn	varchar(11)
sqlstate	varchar(5)

Pruning the recovery history file

The database manager creates entries in the recovery history file for events such as a backup operation, a restore operation, table space creation, and others. When an entry in the recovery history file is no longer relevant, because the associated recovery objects would no longer be needed to recover the database, you might want to remove, or *prune*, those entries from the recovery history file.

You can prune the entries in the recovery history file using the following methods:
- Invoke the PRUNE HISTORY command
- Call the db2Prune API
- Call the ADMIN_CMD procedure with the PRUNE_HISTORY parameter

When you use one of these methods to prune the recovery history file, the database manager removes (prunes) entries from the recovery history file that are older than a timestamp you specify.

If a recovery history file entry matches the criteria you specify for pruning, but that entry would still be needed for a recovery of the database, the database manager will not prune the entry unless you use the WITH FORCE parameter or the DB2PRUNE_OPTION_FORCE flag.

If you use the AND DELETE parameter or the DB2PRUNE_OPTION_DELETE flag, then log files associated with pruned entries will be deleted as well.

If you set the AUTO_DEL_REC_OBJ database configuration parameter to ON, and you use the AND DELETE parameter or the DB2PRUNE_OPTION_DELETE flag, then log files, backup images, and load copy images associated with pruned entries will be deleted.

Automating recovery history file pruning

You can configure the database manager to automatically prune and update the status of recovery history file entries.

You can manually update the status of entries in the recovery history file using the UPDATE HISTORY command, the db2HistoryUpdate API, or the ADMIN_CMD procedure with the ″UPDATE_HISTORY″ parameter. You can use the PRUNE HISTORY command, the db2Prune API, or the ADMIN_CMD procedure with the ″PRUNE_HISTORY″ parameter to manually remove, or prune, entries from the recovery history file. However, it is recommended that you configure the database manager to manage the recovery history file instead of updating and pruning the recovery history file manually.

The database manager automatically updates and prunes recovery history file entries at the following times:

- After a full (non-incremental) database backup operation or full (non-incremental) table space operation completes successfully
- After a database restore operation, where a rollforward operation is not required, completes successfully
- After a database rollforward operation completes successfully

During automated pruning, the database manager performs two operations:

1. Updates the status of the recovery history file entries
2. Prunes expired recovery history file entries

The database manager updates the recovery history file entries in the following way:

- All active database backup images that are no longer needed are marked as expired.
- All database backup images that are marked as inactive and that were taken prior to the point at which an expired database backup was taken are also marked as expired. All associated inactive table space backup images and load backup copies are also marked as expired.
- If an active database backup image is restored, but it is not the most recent database backup recorded in the history file, any subsequent database backup images belonging to the same log sequence are marked as inactive.
- If an inactive database backup image is restored, any inactive database backups belonging to the current log sequence are marked as active again. All active database backup images that are no longer in the current log sequence are marked as inactive.
- Any database or table space backup image that does not correspond to the current log sequence, also called the current log chain, is marked inactive.

  The current log sequence is determined by the database backup image that has been restored, and the log files that have been processed. Once a database backup image is restored, all subsequent database backup images become inactive, because the restored image begins a new log chain. (This is true if the backup image was restored without rolling forward. If a rollforward operation

has occurred, all database backups that were taken after the break in the log chain are marked as inactive. It is conceivable that an older database backup image will have to be restored because the rollforward utility has gone through the log sequence containing a damaged current backup image.)
- A table space-level backup image becomes inactive if, after it is restored, the current state of the database cannot be reached by applying the current log sequence.
- Any entries that have a status of do_not_delete are not pruned, and their associated log files, backup images, and load copy images are not deleted.
- When a database is upgraded, all online database backup entries and all online or offline table space backup entries in the history file are marked as expired, so that these entries are not selected by automatic rebuild as images required for rebuilding. Load copy images and log archive entries are also marked as expired, since these types of entries cannot be used for recovery purposes

The following database configuration parameters control which entries the database manager prunes:

num_db_backups
specifies the number of database backups to retain for a database

rec_his_retentn
specifies the number of days that historical information on backups will be retained

auto_del_rec_obj
specifies whether the database manager should delete log files, backup images, and load copy images that are associated with recovery history file entries that are pruned

To configure the database manager to automatically manage the recovery history file, set the following configuration parameters:
- num_db_backups
- rec_his_retentn
- auto_del_rec_obj

When auto_del_rec_obj is set to ON, and whenever there are more successful database backup entries than the num_db_backups configuration parameter, then the database manager will automatically prune recovery history file entries that are older than rec_his_retentn.

Protecting recovery history file entries from being pruned

You can prevent key recovery history file entries from being pruned, and associated recovery objects from being deleted by setting the status of the recovery history files entries to do_not_delete.

You can remove (prune) recovery history file entries using the PRUNE HISTORY command, the ADMIN_CMD procedure with PRUNE_HISTORY, or the db2Prune API. If you use the AND DELETE parameter with PRUNE HISTORY, or the DB2PRUNE_OPTION_DELETE flag with db2Prune, and the AUTO_DEL_REC_OBJ database configuration parameter is set to ON, then the associated recovery objects will also be physically deleted.

The database manager also prunes the recovery history file entries as part of automated pruning. If the AUTO_DEL_REC_OBJ database configuration parameter is set to ON, the database manager will delete the recovery objects associated with any entries that are pruned.

To protect key recovery history file entries and associated recovery objects:

Use the UPDATE HISTORY command, the db2HistoryUpdate API, or the ADMIN_CMD procedure with "UPDATE_HISTORY" to set the status for key recovery file entries to do_no_delete
There are three ways to prune entries marked do_not_delete:
- Invoke the PRUNE HISTORY command with the WITH FORCE option
- Call the ADMIN_CMD procedure with PRUNE HISTORY and WITH FORCE option
- Call the db2Prune API with the DB2_PRUNE_OPTION_FORCE iOption

Those entries that are marked do_not_delete will never be pruned from the recovery history file unless you perform one of these three procedures.
Restrictions:
- You can set the status of only backup images, load copy images, and log files to do_not_delete.
- The status of a backup entry is not propagated to load copy images, non-incremental backups, or log files related to that backup operation. If you want to save a particular database backup entry and its related log file entries, you must set the status for the database backup entry and the entry for each related log file.

Chapter 9. Managing recovery objects

As you regularly backup your database, you might accumulate very large database backup images, and many database logs and load copy images. The IBM Data Server database manager can simplify managing these recovery objects.

Storing recovery objects can consume great amounts of storage space. Once subsequent backup operations are run, you can delete the older recovery objects because they are no longer needed to restore the database. However, removing the older recovery objects can be time consuming. Also, while you are deleting the older recovery objects, you might accidentally damage recovery objects that are still needed.

There are two ways to use the database manager to delete recovery objects that are no longer required to restore the database:

- You can invoke the PRUNE HISTORY command with the AND DELETE parameter, or call the db2Prune API with the DB2PRUNE_OPTION_DELETE flag.
- You can configure the database manager to automatically delete unneeded recovery objects.

Deleting database recovery objects using the PRUNE HISTORY command or the db2Prune API

You can use the AUTO_DEL_REC_OBJ database configuration parameter and the PRUNE HISTORY command or the db2Prune API to delete recovery objects.

When you invoke the PRUNE HISTORY command, or call the db2Prune API, the IBM Data Server database manager does the following:

- Prunes entries from the recovery history file that do not have the status DB2HISTORY_STATUS_DO_NOT_DEL

When you invoke the PRUNE HISTORY command with the AND DELETE parameter, or when you call the db2Prune API with the DB2PRUNE_OPTION_DELETE flag, the database manager does the following:

- Prunes entries from the recovery history file that are older than a timestamp you specify and that do not have the status DB2HISTORY_STATUS_DO_NOT_DEL
- Deletes the physical log files associated with the pruned entries

If you set the AUTO_DEL_REC_OBJ database configuration parameter to ON, then when you invoke the PRUNE HISTORY command with the AND DELETE parameter, or when you call the db2Prune API with the DB2PRUNE_OPTION_DELETE flag, the database manager does the following:

- Prunes entries from the recovery history file that do not have the status DB2HISTORY_STATUS_DO_NOT_DEL
- Deletes the physical log files associated with the pruned entries
- Deletes the backup images associated with the pruned entries
- Deletes the load copy images associated with the pruned entries

To delete unneeded recovery objects:

1. Set the AUTO_DEL_REC_OBJ database configuration parameter to ON.
2. Invoke the PRUNE HISTORY command with the AND DELETE parameter, or call the db2Prune API with the DB2PRUNE_OPTION_DELETE flag.

Automating database recovery object management

You can use the AUTO_DEL_REC_OBJ database configuration parameter and automated recovery history file pruning to configure the IBM Data Server database manager to automatically delete unneeded recovery objects after every full database backup operation.

After every successful full (non-incremental) database backup operation, the database manager will prune the recovery history file according to the value of the num_db_backup and rec_his_retentn configuration parameters:

- If there are more database backup entries in the recovery history file than the value of the num_db_backup configuration parameter, the database manager will prune those entries from the recovery history file that are older than the value of the rec_his_retentn configuration parameter and that do not have the status DB2HISTORY_STATUS_DO_NOT_DEL.

If you set the AUTO_DEL_REC_OBJ database configuration parameter to ON, then the database manager will do the following in addition to pruning entries from the recovery history file:

- Delete the physical log files associated with the pruned entries
- Delete the backup images associated with the pruned entries
- Delete the load copy images associated with the pruned entries

If there are no full database backup images available for consideration in the current recovery history (perhaps none were ever taken), then images older than the range of time specified by REC_HIS_RETENTN will be deleted.

If the database manager is unable to delete a file because the file is no longer at the location listed in the recovery history file, then the database manager will prune the history entry.

If the database manager is unable to delete a file because of a communication error between the database manager and the storage manager or device, then the database manager will not prune the history file entry. When the error is resolved, the file can be deleted during the next automated prune.

To configure the database manager to automatically delete unneeded recovery objects:

1. Set the AUTO_DEL_REC_OBJ database configuration parameter to ON
2. Set the rec_his_retentn and num_db_backups configuration parameters to enable automated recovery history file pruning.

Protecting recovery objects from being deleted

Automated recovery object management saves administration time and storage space. However, you might want to prevent certain recovery objects from being automatically deleted. You can prevent key recovery objects from being deleted by setting the status of the associated recovery history files entries to do_not_delete.

About this task

If you set the AUTO_DEL_REC_OBJ database configuration parameter to ON, then recovery objects get deleted when their associated recovery history file entries get pruned. Recovery history file entries get pruned when one of the following happens:

- You invoke the PRUNE HISTORY command with the AND DELETE parameter
- You call the db2Prune API with the DB2PRUNE_OPTION_DELETE flag
- The database manager automatically prunes the recovery history file, which happens after every successful table space or database full backup.

Whether you invoke the PRUNE HISTORY command, call the db2Prune API, or configure the database manager to automatically prune the entries in the recovery history file, entries that are marked do_not_delete will not be pruned, and the associated recovery objects will not be deleted.

Procedure

Use the UPDATE HISTORY command to set the status for associated recovery file entries to do_no_delete.
Restrictions:

- You can set the status of only backup images, load copy images, and log files to do_not_delete.
- The status of a backup entry is not propagated to log files, load copy images, or non-incremental backups related to that backup operation. If you want to save a particular database backup entry and its related log file entries, you must set the status for the database backup entry and the entry for each related log file.

Managing snapshot backup objects

You must use the db2acsutil command to manage snapshot backup objects. Do not move or delete snapshot backup objects using file system utilities.

Before you begin

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage® SAN Volume Controller
- IBM System Storage™ DS6000™
- IBM System Storage DS8000®
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS series

Before you can manage snapshot backup objects, you must enable DB2 Advanced Copy Services (ACS). See: "Enabling DB2 Advanced Copy Services (ACS)" on page 291.

Restrictions

The db2acsutil command is currently only supported on AIX and Linux.

Procedure

1. To list available snapshot backup objects, use the **QUERY** parameter.

For example, to list available snapshot backup objects for the database manager instance named `dbminst1`, use the following syntax:

```
db2acsutil query instance dbminst1
```

2. To check the progress of a given snapshot backup operation, use the **STATUS** parameter.

   For example, to see the progress of snapshot backup operations that might be currently running on a database called `database1`, use the following syntax:

   ```
   db2acsutil query status db database1
   ```

3. To delete a particular snapshot backup object, use the **DELETE** parameter.

   For example, to delete all snapshot backup objects for the database called `database1` that are older then 10 days, use the following syntax:

   ```
   db2acsutil delete older than 10 days ago db database1
   ```

Chapter 10. Monitoring the progress of backup, restore and recovery operations

You can use the LIST UTILITIES command to monitor backup, restore, and rollforward operations on a database.

To use progress monitoring for backup, restore and recovery operations:

Issue the LIST UTILITIES command and specify the SHOW DETAIL option.

```
list utilities show detail
```

The following is an example of the output for monitoring the performance of an offline database backup operation:

```
LIST UTILITIES SHOW DETAIL

ID                          = 2
Type                        = BACKUP
Database Name               = SAMPLE
Description                 = offline db
Start Time                  = 10/30/2003 12:55:31.786115
Throttling:
Priority                    = Unthrottled
Progress Monitoring:
Estimated Percentage Complete    = 41
      Total Work Units     = 20232453 bytes
      Completed Work Units = 230637 bytes
      Start Time           = 10/30/2003 12:55:31.786115
```

For backup operations, an initial estimate of the number of bytes to be processed will be specified. As the backup operation progresses the number of bytes to be processed will be updated. The bytes shown does not correspond to the size of the image and should not be used as an estimate for backup image size. The actual image might be much smaller depending on whether it is an incremental or compressed backup.
For restore operations, no initial estimate will be given. Instead UNKNOWN will be specified. As each buffer is read from the image, the actual amount of bytes read will be updated. For automatic incremental restore operations where multiple images might be restored, the progress will be tracked using phases. Each phase represents an image to be restored from the incremental chain. Initially, only one phase will be indicated. After the first image is restored, the total number of phases will be indicated. As each image is restored the number of phases completed will be updated, as will the number of bytes processed.
For crash recovery and rollforward recovery there will be two phases of progress monitoring: FORWARD and BACKWARD. During the FORWARD phase, log files are read and the log records are applied to the database. For crash recovery, the total amount of work is estimated using the starting log sequence number up to the end of the last log file. For rollforward recovery, when this phase begins UNKNOWN will be specified for the total work estimate. The amount of work processed in bytes will be updated as the process continues.
During the BACKWARD phase, any uncommitted changes applied during the FORWARD phase are rolled back. An estimate for the amount of log data to be processed in bytes will be provided. The amount of work processed in bytes will be updated as the process continues.

Table space states

The current status of a table space is reflected by its *state*. The table space states most commonly associated with recovery are:

- *Backup pending*. A table space is put in this state after a point-in-time rollforward operation, or after a load operation with the no copy option. The table space must be backed up before it can be used. (If it is not backed up, the table space cannot be updated, but read-only operations are allowed.)
- *Restore pending*. A table space is put in this state if a rollforward operation on that table space is cancelled, or if a rollforward operation on that table space encounters an unrecoverable error, in which case the table space must be restored and rolled forward again. A table space is also put in this state if, during a restore operation, the table space cannot be restored.
- *Rollforward-in-progress*. A table space is put in this state when a rollforward operation on that table space is in progress. Once the rollforward operation completes successfully, the table space is no longer in rollforward-in-progress state. The table space can also be taken out of this state if the rollforward operation is cancelled.
- *Rollforward pending*. A table space is put in this state after it is restored, or following an input/output (I/O) error. After it is restored, the table space can be rolled forward to the end of the logs or to a point in time. Following an I/O error, the table space must be rolled forward to the end of the logs.

Chapter 11. Backup overview

The simplest form of the DB2 BACKUP DATABASE command requires only that you specify the alias name of the database that you want to back up. For example:

```
db2 backup db sample
```

If the command completes successfully, you will have acquired a new backup image that is located in the path or the directory from which the command was issued. It is located in this directory because the command in this example does not explicitly specify a target location for the backup image. Backup images created by DB2 version 9.5 and later are generated with file mode 600, meaning that on UNIX only the instance owner has read and write privileges and on Windows only members of the DB2ADMNS (and Administrators) group have access to the backup images.

Note: If the DB2 client and server are not located on the same system, DB2 will determine which directory is the current working directory on the client machine and use that as the backup target directory on the server. For this reason, it is recommended that you specify a target directory for the backup image.

Backup images are created at the target location specified when you invoke the backup utility. This location can be:

- A directory (for backups to disk or diskette)
- A device (for backups to tape)
- A Tivoli Storage Manager (TSM) server
- Another vendor's server

The recovery history file is updated automatically with summary information whenever you invoke a database backup operation. This file is created in the same directory as the database configuration file.

If you want to delete old backup images that are no longer required, you can remove the files if the backups are stored as files. If you subsequently run a LIST HISTORY command with the BACKUP option, information about the deleted backup images will also be returned. You must use the PRUNE command to remove those entries from the recovery history file.

If your recovery objects were saved using Tivoli Storage Manager (TSM), you can use the db2adutl utility to query, extract, verify, and delete the recovery objects. On Linux and UNIX, this utility is located in the sqllib/adsm directory, and on Windows operating systems, it is located in sqllib\bin.

On all operating systems, file names for backup images created on disk consist of a concatenation of several elements, separated by periods:

```
DB_alias.Type.Inst_name.NODEnnnn.CATNnnnn.timestamp.Seq_num
```

For example:

```
STAFF.0.DB201.NODE0000.CATN0000.19950922120112.001
```

Note: DB2 Universal Database, Version 8.2.2 and earlier versions used a four-level subdirectory tree when storing backup images on Windows operating systems:

```
DB_alias.Type\Inst_name\NODEnnnn\CATNnnnn\yyyymmdd\hhmmss.Seq_num
```

For example:

```
SAMPLE.0\DB2\NODE0000\CATN0000\20010320\122644.001
```

Database alias
: A 1- to 8-character database alias name that was specified when the backup utility was invoked.

Type
: Type of backup operation, where: `0` represents a full database-level backup, `3` represents a table space-level backup, and `4` represents a backup image generated by the LOAD...COPY TO command.

Instance name
: A 1- to 8-character name of the current instance that is taken from the DB2INSTANCE environment variable.

Node number
: The database partition number. In single partition database environments, this is always `NODE0000`. In partitioned database environments, it is `NODE`*xxxx*, where *xxxx* is the number assigned to the database partition in the db2nodes.cfg file.

Catalog partition number
: The database partition number of the catalog partition for the database. In single partition database environments, this is always `CATN0000`. In partitioned database environments, it is `CATN`*xxxx*, where *xxxx* is the number assigned to the database partition in the `db2nodes.cfg` file.

Time stamp
: A 14-character representation of the date and time at which the backup operation was performed. The time stamp is in the form *yyyymmddhhnnss*, where:

- *yyyy* represents the year (1995 to 9999)
- *mm* represents the month (01 to 12)
- *dd* represents the day of the month (01 to 31)
- *hh* represents the hour (00 to 23)
- *nn* represents the minutes (00 to 59)
- *ss* represents the seconds (00 to 59)

Sequence number
: A 3-digit number used as a file extension.

When a backup image is written to tape:

- File names are not created, but the information described above is stored in the backup header for verification purposes.
- A tape device must be available through the standard operating system interface. In a large partitioned database environment, however, it might not be practical to have a tape device dedicated to each database partition server. You can connect the tape devices to one or more TSM servers, so that access to these tape devices is provided to each database partition server.
- In a partitioned database environment, you can also use products that provide virtual tape device functions, such as REELlibrarian 4.2 or CLIO/S. You can use these products to access the tape device connected to other nodes (database partition servers) through a pseudo tape device. Access to the remote tape device is provided transparently, and the pseudo tape device can be accessed through the standard operating system interface.

You cannot back up a database that is in an unusable state, except when that database is in backup pending state. If any table space is in an abnormal state, you cannot back up the database or that table space, unless it is in backup pending state.

Concurrent backup operations on the same table space are not permitted. Once a backup operation has been initiated on a table space, any subsequent attempts will fail (SQL2048).

If a database or a table space is in a partially restored state because a system crash occurred during the restore operation, you must successfully restore the database or the table space before you can back it up.

A backup operation will fail if a list of the table spaces to be backed up contains the name of a temporary table space.

The backup utility provides concurrency control for multiple processes that are making backup copies of different databases. This concurrency control keeps the backup target devices open until all the backup operations have ended. If an error occurs during a backup operation, and an open container cannot be closed, other backup operations targeting the same drive might receive access errors. To correct such access errors, you must terminate the backup operation that caused the error and disconnect from the target device. If you are using the backup utility for concurrent backup operations to tape, ensure that the processes do not target the same tape.

Displaying backup information

You can use db2ckbkp to display information about existing backup images. This utility allows you to:

- Test the integrity of a backup image and determine whether or not it can be restored.
- Display information that is stored in the backup header.
- Display information about the objects and the log file header in the backup image.

Using backup

Use the BACKUP DATABASE command to take a copy of a database’s data and store it on a different medium in case of failure or damage to the original. You can back up an entire database, database partition, or only selected table spaces.

Before you begin

You do not need to be connected to the database that is to be backed up: the backup database utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the backup operation. If you are connected to a database that is to be backed up, you will be disconnected when the BACKUP DATABASE command is issued and the backup operation will proceed.

The database can be local or remote. The backup image remains on the database server, unless you are using a storage management product such as Tivoli Storage Manager (TSM) or DB2 Advanced Copy Services (ACS).

If you are performing an offline backup and if you have activated the database using the ACTIVATE DATABASE statement, you must deactivate the database before you run the offline backup. If there are active connections to the database, in order to deactivate the database successfully, a user with SYSADM authority must connect to the database and issue the following commands:

```
CONNECT TO database-alias
QUIESCE DATABASE IMMEDIATE FORCE CONNECTIONS;
UNQUIESCE DATABASE;
TERMINATE;
DEACTIVATE DATABASE database-alias
```

In a partitioned database environment, you can use the BACKUP DATABASE command to back up database partitions individually, use the ON DBPARTITIONNUM command parameter to back up several of the database partitions at once, or use the ALL DBPARTITIONNUMS parameter to back up all of the database partitions simultaneously. You can use the LIST NODES command to identify the database partitions that have user tables on them that you might want to back up.

Unless you are using a single system view (SSV) backup, if you are performing an *offline* backup in a partitioned database environment, you should back up the catalog partition separately from all other database partitions. For example, you can back up the catalog partition first, then back up the other database partitions. This is necessary because the backup operation may require an exclusive database connection on the catalog partition, during which the other database partitions cannot connect. If you are performing an *online* backup, all database partitions (including the catalog partition) can be backed up simultaneously or in any order.

You can also use the Command Editor to back up database partitions. Because this approach does not support rollforward recovery, you should back up the database residing on these nodes regularly. You should also keep a copy of the db2nodes.cfg file with any backup copies you take, as protection against possible damage to this file.

On a distributed request system, backup operations apply to the distributed request database and the metadata stored in the database catalog (wrappers, servers, nicknames, and so on). Data source objects (tables and views) are not backed up, unless they are stored in the distributed request database

If a database was created with a previous release of the database manager, and the database has not been upgraded, you must upgrade the database before you can back it up.

About this task

The following restrictions apply to the backup utility:

- A table space backup operation and a table space restore operation cannot be run at the same time, even if different table spaces are involved.
- If you want to be able to do rollforward recovery in a partitioned database environment, you must regularly back up the database on the list of nodes, and you must have at least one backup image of the rest of the nodes in the system (even those that do not contain user data for that database). Two situations require the backed-up image of a database partition at a database partition server that does not contain user data for the database:

  - You added a database partition server to the database system after taking the last backup, and you need to do forward recovery on this database partition server.
  - Point-in-time recovery is used, which requires that all database partitions in the system are in rollforward pending state.
- Online backup operations for DMS table spaces are incompatible with the following operations:
  - load
  - reorganization (online and offline)
  - drop table space
  - table truncation
  - index creation
  - not logged initially (used with the CREATE TABLE and ALTER TABLE statements)
- If you attempt to perform an offline backup of a database that is currently active, you will receive an error. Before you run an offline backup you can make sure the database is not active by issuing the DEACTIVATE DATABASE command.

The backup utility can be invoked through the command line processor (CLP), the Backup Database wizard in the Control Center, by running the ADMIN_CMD procedure with the BACKUP DATABASE parameter, or the *db2Backup* application programming interface (API).

Following is an example of the BACKUP DATABASE command issued through the CLP:

```
db2 backup database sample to c:\DB2Backups
```

Procedure

To open the Backup Database wizard:

1. From the Control Center, expand the object tree until you find the database or table space object that you want to back up.
2. Right-click on the object and select Backup from the pop-up menu. The Backup Database wizard opens. .

What to do next

Detailed information is provided through the contextual help facility within the Control Center

If you performed an offline backup, after the backup completes, you must reactivate the database:

```
ACTIVATE DATABASE sample
```

Performing a snapshot backup

A snapshot backup operation uses the fast copying technology of a storage device to perform the data copying portion of the backup.

Before you begin

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS series

Before you can perform a snapshot backup, you must enable DB2 Advanced Copy Services (ACS). See: "Enabling DB2 Advanced Copy Services (ACS)" on page 291.

Procedure

You can perform a snapshot backup using the BACKUP DATABASE command with the **USE SNAPSHOT** parameter, the db2Backup API with the SQLU_SNAPSHOT_MEDIA media type, or the ADMIN_CMD procedure with the **BACKUP DATABASE** and the **USE SNAPSHOT** parameters:
BACKUP DATABASE command:

```
db2 backup db sample use snapshot
```

ADMIN_CMD procedure with **BACKUP DATABASE** parameter:

```
CALL SYSPROC.ADMIN_CMD
   ('backup db sample use snapshot')
```

db2Backup API

```
int sampleBackupFunction( char dbAlias[],
                          char user[],
                          char pswd[],
                          char workingPath[] )
{
  db2MediaListStruct mediaListStruct = { 0 };

  mediaListStruct.locations = &workingPath;
  mediaListStruct.numLocations = 1;
  mediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;


  db2BackupStruct backupStruct = { 0 };

  backupStruct.piDBAlias = dbAlias;
  backupStruct.piUsername = user;
  backupStruct.piPassword = pswd;
  backupStruct.piVendorOptions = NULL;
  backupStruct.piMediaList = &mediaListStruct;

  db2Backup(db2Version950, &backupStruct, &sqlca);

  return 0;
}
```

Results

After performing a snapshot backup, you can restore from the snapshot backup.

Using a split mirror as a backup image

Use the following procedure to create a split mirror of a primary database for use as a backup image. This procedure can be used instead of performing backup database operations on the primary database.

To use a split mirror as a “backup image”, follow these steps:

1. Suspend I/O on the primary database:
   ```
   db2 set write suspend for database
   ```
   While the database is suspended, you should not be running other utilities or tools. You should only be making a copy of the database.
2. Use appropriate operating system-level commands to split the mirror or mirrors from the primary database.

   Note: Ensure that you copy the entire database directory including the volume directory. You must also copy the log directory and any container directories that exist outside the database directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.
3. Resume I/O on the primary database:
   ```
   db2 set write resume for database
   ```
4. A failure occurs on the primary system, necessitating a restore from backup.
5. Stop the primary database instance:
   ```
   db2stop
   ```
6. Use operating system-level commands to copy the split-off data over the primary system. **Do not copy the split-off log files**, because the primary logs will be needed for rollforward recovery.
7. Start the primary database instance:
   ```
   db2start
   ```
8. Initialize the primary database:
   ```
   db2inidb database_alias as mirror
   ```
9. Roll the primary database forward to the end of the logs or to a point-in-time and stop.

Backing up to tape

When you back up your database or table space, you must correctly set your block size and your buffer size. This is particularly true if you are using a variable block size (on AIX, for example, if the block size has been set to zero).

There is a restriction on the number of fixed block sizes that can be used when backing up. This restriction exists because DB2 writes out the backup image header as a 4-KB block. The only fixed block sizes DB2 supports are 512, 1024, 2048, and 4096 bytes. If you are using a fixed block size, you can specify any backup buffer size. However, you might find that your backup operation will not complete successfully if the fixed block size is not one of the sizes that DB2 supports.

If your database is large, using a fixed block size means that your backup operations might take a long time to complete. You might want to consider using a variable block size.

Note: 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.

Before a tape device can be used on a Windows operating system, the following command must be issued:

```
db2 initialize tape on <device> using <blksize>
```

Where:

<device>
: is a valid tape device name. The default on Windows operating systems is `\\.\TAPE0`.

<blksize>
: is the blocking factor for the tape. It must be a factor or multiple of 4096. The default value is the default block size for the device.

Restoring from a backup image with variable block size might return an error. If this happens, you might need to rewrite the image using an appropriate block size. Following is an example on AIX:

```
tctl -b 0 -Bn -f /dev/rmt0 read > backup_filename.file
dd if=backup_filename.file of=/dev/rmt0 obs=4096 conv=sync
```

The backup image is dumped to a file called backup_filename.file. The dd command dumps the image back onto tape, using a block size of 4096.

There is a problem with this approach if the image is too large to dump to a file. One possible solution is to use the dd command to dump the image from one tape device to another. This will work as long as the image does not span more than one tape. When using two tape devices, the dd command is:

```
dd if=/dev/rmt1 of=/dev/rmt0 obs=4096
```

If using two tape devices is not possible, you might be able to dump the image to a raw device using the dd command, and then to dump the image from the raw device to tape. The problem with this approach is that the dd command *must* keep track of the number of blocks dumped to the raw device. This number must be specified when the image is moved back to tape. If the dd command is used to dump the image from the raw device to tape, the command dumps the entire contents of the raw device to tape. The dd utility cannot determine how much of the raw device is used to hold the image.

When using the backup utility, you will need to know the maximum block size limit for your tape devices. Here are some examples:

Device	**Attachment**	**Block Size Limit**	**DB2 Buffer Size Limit (in 4-KB pages)**
8 mm	scsi	131,072	32
3420	s370	65,536	16

Device	Attachment	Block Size Limit	DB2 Buffer Size Limit (in 4-KB pages)
3480	s370	61 440	15
3490	s370	61 440	15
3490E	s370	65,536	16
7332 (4 mm)[1]	scsi	262,144	64
3490e	scsi	262,144	64
3590[2]	scsi	2,097,152	512
3570 (magstar MP)		262,144	64

Note:

1. The 7332 does not implement a block size limit. 256 KB is simply a suggested value. Block size limit is imposed by the parent adapter.
2. While the 3590 does support a 2-MB block size, you could experiment with lower values (like 256 KB), provided the performance is adequate for your needs.
3. For information about your device limit, check your device documentation or consult with the device vendor.

Verifying the compatibility of your tape device

On UNIX, Linux, and AIX operating systems only, to determine whether your tape device is supported for backing up your DB2 databases, perform the following procedure:

As the database manager instance owner, run the operating system command dd to read from or write to your tape device. If the dd command succeeds, then you can back up your DB2 databases using your tape device.

Backing up to named pipes

Support is now available for database backup to (and database restore from) local named pipes on UNIX based systems.

Both the writer and the reader of the named pipe must be on the same machine. The pipe must exist and be located on a local file system. Because the named pipe is treated as a local device, there is no need to specify that the target is a named pipe.

Following is an AIX example:

1. Create a named pipe:
   ```
   mkfifo /u/dmcinnis/mypipe
   ```
2. If this backup image is going to be used by the restore utility, the restore operation must be invoked *before* the backup operation, so that it will not miss any data:
   ```
   db2 restore db sample into mynewdb from /u/dmcinnis/mypipe
   ```
3. Use this pipe as the target for a database backup operation:
   ```
   db2 backup db sample to /u/dmcinnis/mypipe
   ```

Backing up partitioned databases

Backing up a database in a partitioned database environment can pose difficulties such as: tracking the success of the backup of each database partition; managing the multiple log files and backup images; and ensuring the log files and backup images for all the database partitions span the minimum recovery time required to restore the database. Using a single system view (SSV) backup is the easiest way to back up a partitioned database.

There are three possible ways to back up a database in a partitioned database environment:

- Back up each database partition one at a time using the BACKUP DATABASE command, the BACKUP DATABASE command with the ADMIN_CMD procedure, or the db2Backup API function
- Use the db2_all command with the BACKUP DATABASE command to back up all the database partitions that you specify
- Run a single system view (SSV) backup to back up some or all of the database partitions simultaneously

Backing up each database partition one at a time is time-consuming and error-prone. Backing up all the partitions using the db2_all command is easier than backing up each database partition individually because you generally only have to make one command call. However, when you use db2_all to back up a partitioned database, you sometimes still have to make multiple calls to db2_all because the database partition containing the catalog can not be backed up simultaneously with non-catalog database partitions. Whether you back up each database partition one at a time or use db2_all, managing backup images created using either of these methods is difficult because the timestamp for each database partition's backup image will be different, and coordinating the minimum recovery time across the database partitions' backup images is difficult as well.

For the reasons mentioned above, the recommended way to back up a database in a partitioned database environment is to use a SSV backup.

To back up some or all of the database partitions of a partitioned database simultaneously using a SSV backup:

1. Optional: allow the database to remain online, or take the database offline.

   You can back up a partitioned database while the database is online or offline. If the database is online, the backup utility will acquire shared connections to the other database partitions, so user applications will be able to connect to database partitions while they are being backed up.
2. On the database partition that contains the database catalog, invoke backup with appropriate parameters for partitioned databases.
   - You can call the BACKUP DATABASE command with the ON DBPARTITIONNUMS parameter.
   - You can call the BACKUP DATABASE command with the ON DBPARTITIONNUMS parameter using the ADMIN_CMD procedure.
   - You can call the db2Backup API with the iAllNodeFlag parameter.
3. Optional: include the log files required for recovery with the backup images.

By default, log files are included with backup images if you are performing a SSV backup (that is, if you specify the ON DBPARTITIONNUM parameter). If you do not want log files to be included with the backup images, use the EXCLUDE LOGS command parameter when you run the backup. Log files are excluded from the backup image by default for non-SSV backups.

See the following topic for more information: "Including log files with a backup image" on page 142.

4. Optional: delete previous backup images. The method you use to delete old backup images depends on how you store the backup images. For example, if you store the backup images to disk, you can delete the files; if you store the backup images using Tivoli storage manager, you can use the db2adutl utility to delete the backup images. If you are using DB2 Advanced Copy Services (ACS), you can use the db2acsutil to delete snapshot backup objects.

Backing up partitioned tables using IBM Tivoli Space Manager Hierarchical Storage Management

The Tivoli Space Manager Hierarchical Storage Manager (HSM) client program automatically migrates eligible files to secondary storage to maintain specific levels of free space on local file systems.

With table partitioning, table data is divided across multiple storage objects called data partitions. HSM supports the backup of individual data partitions to secondary storage.

When using SMS table spaces, each data partition range is represented as a file in the corresponding directory. Therefore, it is very easy to migrate individual ranges of data (data partitions) to secondary storage.

When using DMS table spaces, each container is represented as a file. In this case, infrequently accessed ranges should be stored in their own table space. When you issue a CREATE TABLE statement using the EVERY clause, use the NO CYCLE clause to ensure that the number of table spaces listed in the table level IN clause match the number of data partitions being created. This is demonstrated in the following example:

Example 1

```
CREATE TABLE t1 (c INT) IN tbsp1, tbsp2, tbsp3 NO CYCLE
  PARTITION BY RANGE(c)
  (STARTING FROM 2 ENDING ( 6 EVERY 2);
```

Enabling automatic backup

A database can become unusable due to a wide variety of hardware or software failures. Ensuring that you have a recent, full backup of your database is an integral part of planning and implementing a disaster recovery strategy for your system. Use automatic database backup as part of your disaster recovery strategy to enable DB2 to back up your database both properly and regularly.

You can configure automatic backup using the graphical user interface tools, the command line interface, or the AUTOMAINT_SET_POLICY system stored procedure.

- To configure automatic backup using the graphical user interface tools:
  1. Open the Configure Automatic Maintenance wizard either from the Control Center by right-clicking on a database object or from the Health Center by

right-clicking on the database instance that you want to configure for automatic backup. Select **Configure Automatic Maintenance** from the pop-up window.

2. Within this wizard, you can enable automatic backup and specify a maintenance window for the execution of the BACKUP utility.

- To configure automatic backup using the command line interface, set each of the following configuration parameters to ON:
  - AUTO_MAINT
  - AUTO_DB_BACKUP
- To configure automatic backup using the AUTOMAINT_SET_POLICY system stored procedure:
  1. Create configuration XML input specifying details like backup media, whether the backup should be online or offline, and frequency of the backup.

     You can copy the contents of the sample file called DB2DefaultAutoBackupPolicy.xml in the SQLLIB/samples/automaintcfg directory and modify the XML to satisfy your configuration requirements.
  2. Optional: create an XML input file containing your configuration XML input.
  3. Call AUTOMAINT_SET_POLICY with the following parameters:
     - maintenance type: AutoBackup
     - configuration XML input: either a BLOB containing your configuration XML input text; or the name of the file containing your configuration XML input.

  See the topic "Configuring an automated maintenance policy using SYSPROC.AUTOMAINT_SET_POLICY or SYSPROC.AUTOMAINT_SET_POLICYFILE" on page 55 for more information about using the AUTOMAINT_SET_POLICY system stored procedure.

Automatic database backup

A database may become unusable due to a wide variety of hardware or software failures. Automatic database backup simplifies database backup management tasks for the DBA by always ensuring that a recent full backup of the database is performed as needed. It determines the need to perform a backup operation based on one or more of the following measures:

- You have never completed a full database backup
- The time elapsed since the last full backup is more than a specified number of hours
- The transaction log space consumed since the last backup is more than a specified number of 4 KB pages (in archive logging mode only).

Protect your data by planning and implementing a disaster recovery strategy for your system. If suitable to your needs, you may incorporate the automatic database backup feature as part of your backup and recovery strategy.

If the database is enabled for roll-forward recovery (archive logging), then automatic database backup can be enabled for either online or offline backup. Otherwise, only offline backup is available. Automatic database backup supports disk, tape, Tivoli Storage Manager (TSM), and vendor DLL media types.

Through the Configure Automatic Maintenance wizard in the Control Center or Health Center, you can configure:

- The requested time or number of log pages between backups
- The backup media
- Whether it will be an online or offline backup.

If backup to disk is selected, the automatic backup feature will regularly delete backup images from the directory specified in the Configure Automatic Maintenance wizard. Only the most recent backup image is guaranteed to be available at any given time. It is recommended that this directory be kept exclusively for the automatic backup feature and not be used to store other backup images.

The automatic database backup feature can be enabled or disabled by using the auto_db_backup and auto_maint database configuration parameters. In a partitioned database environment, the automatic database backup runs on each database partition if the database configuration parameters are enabled on that database partition.

You can also configure automatic backup using one of the system stored procedures called AUTOMAINT_SET_POLICY and AUTOMAINT_SET_POLICYFILE.

Optimizing backup performance

When you perform a backup operation, DB2 will automatically choose an optimal value for the number of buffers, the buffer size and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available, and the database configuration. Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the UTIL_HEAP_SZ configuration parameter. The objective is to minimize the time it takes to complete a backup operation. Unless you explicitly enter a value for the following BACKUP DATABASE command parameters, DB2 will select one for them:

- WITH num-buffers BUFFERS
- PARALLELISM n
- BUFFER buffer-size

If the number of buffers and the buffer size are not specified, resulting in DB2 setting the values, it should have minimal effect on large databases. However, for small databases, it can cause a large percentage increase in backup image size. Even if the last data buffer written to disk contains little data, the full buffer is written to the image anyway. In a small database, this means that a considerable percentage of the image size might be empty.

You can also choose to do any of the following to reduce the amount of time required to complete a backup operation:

- Specify table space backup.

  You can back up (and subsequently recover) part of a database by using the TABLESPACE option on the BACKUP DATABASE command. This facilitates the management of table data, indexes, and long field or large object (LOB) data in separate table spaces.
- Increase the value of the PARALLELISM parameter on the BACKUP DATABASE command so that it reflects the number of table spaces being backed up.

  The PARALLELISM parameter defines the number of processes or threads that are started to read data from the database and to compress data during a compressed backup operation. Each process or thread is assigned to a specific table space, so there is no benefit to specifying a value for the PARALLELISM parameter that is larger than the number of table spaces being backed up. When it finishes backing up this table space, it requests another. Note, however, that each process or thread requires both memory and CPU overhead.

- Increase the backup buffer size.

  The ideal backup buffer size is a multiple of the table space extent size plus one page. If you have multiple table spaces with different extent sizes, specify a value that is a common multiple of the extent sizes plus one page.
- Increase the number of buffers.

  Use at least twice as many buffers as backup targets (or sessions) to ensure that the backup target devices do not have to wait for data.
- Use multiple target devices.

Privileges, authorities, and authorization required to use backup

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the backup utility.

Compatibility of online backup and other utilities

Some utilities can be run at the same time as an online backup, but others cannot.

The following utilities are compatible with online backup:

- EXPORT
- ONLINE INSPECT

The following utilities are compatible with online backup only under certain circumstances:

- ONLINE CREATE INDEX

  In SMS mode, online index create and online backup will not be compatible due to the ALTER TABLE lock. Online index create acquires it in exclusive mode while online backup acquires it in share.

  In DMS mode, online index create and online backup can run concurrently in most cases. There is a possibility if you have a large number of indexes that the online index create will internally acquire an online backup lock that will conflict with any concurrent online backup.
- ONLINE INDEX REORG

  As with online index create, in SMS mode, online index reorganization is not compatible with online backup due to the ALTER TABLE lock. Online index reorganization acquires it in exclusive mode while online backup acquires it in share. In addition, an online index reorganization operation, quiesces the table before the switch phase and acquires a Z lock, which prevents an online backup. However, the ALTER TABLE lock should prevent an online backup from running concurrently before the Z table lock is acquired.

In DMS mode, online index reorganization and online backup can run concurrently.

In addition, online index reorganization quiesces the table before the switch phase and gets a Z lock, which prevents an online backup.

- REBALANCE

  When online backup and rebalancer are running concurrently, online backup will pause the rebalancer and does not wait for it to complete.

- IMPORT

  The import utility is compatible with online backup except when the IMPORT command is issued with the REPLACE option, in which case, import gets a Z lock on the table and prevents an online backup from running concurrently.

- ALLOW READ ACCESS LOAD

  ALLOW READ ACCESS load operations are not compatible with online backup when the LOAD command is issued with the COPY NO option. In this mode the utilities both modify the table space state, causing one of the utilities to report an error.

  ALLOW READ ACCESS load operations are compatible with online backup when the LOAD command is issued with the COPY YES option, although there might still be some compatibility issues. In SMS mode, the utilities can execute concurrently, but they will hold incompatible table lock modes and consequently might be subject to table lock waits. In DMS mode, the utilities both hold incompatible ″Internal-B″ (OLB) lock modes and might be subject to waits on that lock. If the utilities execute on the same table space concurrently, the load utility might be forced to wait for the backup utility to complete processing of the table space before the load utility can proceed.

- ONLINE TABLE REORG

  The clean up phase of online table reorganization cannot start while an online backup is running. You can pause the table reorganization, if required, to allow the online backup to finish before resuming the online table reorg.

  You can start an online backup of a DMS table space when a table within the same table space is being reorganized online. There might be lock waits associated with the reorganization operation during the truncate phase.

  You cannot start an online backup of an SMS table space when a table within the same table space is being reorganized online. Both operations require an exclusive lock.

- DDLs that require a Z lock (such as ALTER TABLE, DROP TABLE and DROP INDEX)

  Online DMS table space backup is compatible with DDLs that require a Z lock.

  Online SMS table space backup must wait for the Z lock to be released.

- RUNSTATS (allow write and allow read)

  Runstats is compatible with online backup except when the system catalog table space is a SMS table space. This is because if the SYSIBM.SYSTABLES resides in a SMS table space, then runstats and online backup will hold incompatible table lock on the table, causing lock waits.

The following utilities are not compatible with online backup:

- REORG TABLE
- RESTORE
- ROLLFORWARD
- ONLINE BACKUP

- ALLOW NO ACCESS LOAD
- SET WRITE

Backup examples

Example 1

In the following example database SAMPLE is backed up to a TSM server using 2 concurrent TSM client sessions. The backup utility will compute the optimal number of buffers. The optimal size of the buffers, in 4 KB pages, is automatically calculated based on the amount of memory and the number of target devices that are available. The parallelism setting is also automatically calculated and is based on the number or processors available and the number of table spaces to be backed up.

```
db2 backup database sample use tsm open 2 sessions with 4 buffers

db2 backup database payroll tablespace (syscatspace, userspace1) to
   /dev/rmt0, /dev/rmt1 with 8 buffers without prompting
```

Example 2

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) db2 backup db kdr use tsm
(Mon) db2 backup db kdr online incremental delta use tsm
(Tue) db2 backup db kdr online incremental delta use tsm
(Wed) db2 backup db kdr online incremental use tsm
(Thu) db2 backup db kdr online incremental delta use tsm
(Fri) db2 backup db kdr online incremental delta use tsm
(Sat) db2 backup db kdr online incremental use tsm
```

Example 3

To initiate a backup operation to a tape device in a Windows environment, issue:

```
db2 backup database sample to \\.\tape0
```

Chapter 12. Recover overview

The recover utility performs the necessary restore and rollforward operations to recover a database to a specified time, based on information found in the recovery history file. When you use this utility, you specify that the database be recovered to a point-in-time or to the end of the log files. The utility will then select the best suitable backup image and perform the recovery operations.

The recover utility does not support the following RESTORE DATABASE command options:

- TABLESPACE tablespace-name. Table space restore operations are not supported.
- INCREMENTAL. Incremental restore operations are not supported.
- OPEN num-sessions SESSIONS. You cannot indicate the number of I/O sessions that are to be used with TSM or another vendor product.
- BUFFER buffer-size. You cannot set the size of the buffer used for the restore operation.
- DLREPORT filename. You cannot specify a file name for reporting files that become unlinked.
- WITHOUT ROLLING FORWARD. You cannot specify that the database is not to be placed in rollforward pending state after a successful restore operation.
- PARALLELISM n. You cannot indicate the degree of parallelism for the restore operation.
- WITHOUT PROMPTING. You cannot specify that a restore operation is to run unattended

In addition, the recover utility does not allow you to specify any of the REBUILD options. However, the recovery utility will automatically use the appropriate REBUILD option if it cannot locate any database backup images based on the information in the recovery history file.

Recovering data

Use the RECOVER DATABASE command to recover a database to a specified time, using information found in the recovery history file.

If you issue the RECOVER DATABASE command following an incomplete recover operation that ended during the rollforward phase, the recover utility will attempt to continue the previous recover operation, without redoing the restore phase. If you want to force the recover utility to redo the restore phase, issue the RECOVER DATABASE command with the RESTART option to force the recover utility to ignore any prior recover operation that failed to complete. If you are using the application programming interface (API), specify the caller action DB2RECOVER_RESTART for the iRecoverAction field to force the recover utility to redo the restore phase.

If the RECOVER DATABASE command is interrupted during the restore phase, it cannot be continued. You need to reissue the RECOVER DATABASE command.

You should not be connected to the database that is to be recovered: the recover database utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the recover operation.

The database can be local or remote.

You can invoke the recover utility through the command line processor (CLP) or the db2Recover application programming interface (API).

The following example shows how to use the RECOVER DATABASE command through the CLP:

```
   db2 recover db sample
```

Note: In a partitioned database environment, the recover utility must be invoked from the catalog partition of the database.

Recovering data using db2adutl

The examples that follow show how to perform cross-node recovery using the db2adutl command, and the *logarchopt1* and *vendoropt* database configuration parameters.

For the following examples, computer 1 is called `bar` and is running AIX. The owner of this machine is `roecken`. The database on `bar` is called `zample`. Computer 2 is called `dps`. This machine is also running AIX, and is owned by `regress9`.

PASSWORDACCESS = generate

Computer 1

1. Set up the database for log archiving to TSM. Update the database configuration parameter *logarchmeth1* for the `zample` database:
   ```
   bar:/home/roecken> db2 update db cfg for zample using LOGARCHMETH1 tsm
   ```
   The following information is returned:
   ```
   DB20000I  The UPDATE DATABASE CONFIGURATION command completed successfully.
   ```
   Note: Before updating the database configuration, you might have to take an offline backup of the database.
2. Force off applications:
   ```
   db2 force applications all
   ```
3. Verify that all applications have been forced off:
   ```
   db2 list applications
   ```
   You should receive a message that says no data was returned.

   Note: In a partitioned database environment, you must perform this step on all database partitions.
4. Take a backup of the database:
   ```
   db2 backup db zample use tsm
   ```
   Information similar to the following is returned:
   ```
   Backup successful. The timestamp for this backup image is : 20040216151025
   ```
   Note: In a partitioned database environment, you must perform this step on all database partitions. The order in which you perform this step on the database partitions differs depending on whether you are performing an online backup or an offline backup. For more information, see “Using backup” on page 199.
5. Connect to the `zample` database, then create a table in it.

6. Load data into the new table. In this example, the table is called `a`, and the data is being loaded from a delimited ASCII file called `mr`. The COPY YES option is specified to make a copy of the data that is loaded, and the USE TSM option specifies that the copy of the data is stored on Tivoli Storage Manager.

   Note: You can only specify the COPY YES option if the database is enabled for rollforward recovery; that is, the *logarchmeth1* database configuration parameter must be set to either USEREXIT or LOGRETAIN.

   ```
   bar:/home/roecken> db2 load from mr of del modified by noheader replace
      into a copy yes use tsm
   ```

   The utility returns a series of messages to indicate its progress:

   ```
   SQL3109N  The utility is beginning to load data from file "/home/roecken/mr".

   SQL3500W  The utility is beginning the "LOAD" phase at time "02/16/2004
   15:12:13.392633".

   SQL3519W  Begin Load Consistency Point. Input record count = "0".

   SQL3520W  Load Consistency Point was successful.

   SQL3110N  The utility has completed processing.  "1" rows were read from the
   input file.

   SQL3519W  Begin Load Consistency Point. Input record count = "1".

   SQL3520W  Load Consistency Point was successful.

   SQL3515W  The utility has finished the "LOAD" phase at time "02/16/2004
   15:12:13.445718".


   Number of rows read         = 1
   Number of rows skipped      = 0
   Number of rows loaded       = 1
   Number of rows rejected     = 0
   Number of rows deleted      = 0
   Number of rows committed    = 1
   ```

   There should now be one backup image, one load copy image and one log file on TSM. A query on the `zample` database can be run as follows:

   ```
   bar:/home/roecken/sqllib/adsm> db2adutl query db zample
   ```

   The following information is returned:

   ```
   Retrieving FULL DATABASE BACKUP information.
       1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 0
       Sessions: 1


   Retrieving INCREMENTAL DATABASE BACKUP information.
     No INCREMENTAL DATABASE BACKUP images found for ZAMPLE


   Retrieving DELTA DATABASE BACKUP information.
     No DELTA DATABASE BACKUP images found for ZAMPLE


   Retrieving TABLESPACE BACKUP information.
     No TABLESPACE BACKUP images found for ZAMPLE


   Retrieving INCREMENTAL TABLESPACE BACKUP information.
     No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE
   ```

```
Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE


Retrieving LOAD COPY information.
    1 Time: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

7. To enable cross-node recovery, another node and account must be given access to the objects on the bar computer. In this example, access is given to the node dps and the user regress9.

   ```
   bar:/home/roecken/sqllib/adsm> db2adutl grant user regress9
      on nodename dps for db zample
   ```

   The following information is returned:

   ```
   Successfully added permissions for regress9 to access ZAMPLE on node dps.
   ```

   To query the results of the db2adutl grant operation, issue the following command:

   ```
   bar:/home/roecken/sqllib/adsm> db2adutl queryaccess
   ```

   The following information is returned:

   ```
   Node                Username            Database Name   Type
   --------------------------------------------------------------
   DPS                 regress9            ZAMPLE          A
   --------------------------------------------------------------
    Access Types:  B - backup images   L - logs   A - both
   ```

PASSWORDACCESS = generate environment

Computer 2

Computer 2, dps, is not yet set up. A db2adutl query on dps for the zample database returns the following results:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample
--- Database directory is empty ---
Warning: There are no file spaces created by DB2 on the ADSM server
Warning: No DB2 backup images found in ADSM for any alias.


dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename
    bar owner roecken
--- Database directory is empty ---

Query for database ZAMPLE


Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 0
    Sessions: 1


Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE


Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE


Retrieving TABLESPACE BACKUP information.
```

```
  No TABLESPACE BACKUP images found for ZAMPLE


Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE


Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE


Retrieving LOAD COPY information.
    1 Time: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

The `zample` database does not yet exist on the `dps` computer.

1. Restore the `zample` database to the `dps` computer:

   ```
   dps:/home/regress9> db2 restore db zample use tsm options
   "'-fromnode=bar -fromowner=roecken'" without prompting
   ```

   The following information is returned:

   ```
   DB20000I  The RESTORE DATABASE command completed successfully.
   ```

   Note: If the `zample` database already existed on `dps`, the OPTIONS parameter would be omitted, and the database configuration parameter *vendoropt* would be used. This configuration parameter overrides the OPTIONS parameter for a backup or restore operation.

   A rollforward operation on the `zample` database will fail because the rollforward utility cannot find the log files. A rollforward operation such as the following:

   ```
   dps:/home/regress9> db2 rollforward db zample to end of logs and stop
   ```

   Returns the following error:

   ```
   SQL4970N  Roll-forward recovery on database "ZAMPLE" cannot reach the
   specified stop point (end-of-log or point-in-time) because of missing log
   file(s) on node(s) "0".
   ```

2. To force the rollforward utility to look for log files on another machine, you must configure the proper *logarchopt* value, in this situation the *logarchopt1* database configuration parameter:

   ```
   dps:/home/regress9> db2 update db cfg for zample using logarchopt1
   "'-fromnode=bar -fromowner=roecken'"
   ```

3. For the rollforward utility to be able to use the load copy images, you must also set the *vendoropt* database configuration parameter:

   ```
   dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
   "'-fromnode=bar -fromowner=roecken'"
   ```

4. The `zample` database can now be rolled forward::

   ```
   dps:/home/regress9> db2 rollforward db zample to end of logs and stop
   ```

   The following information is returned:

   ```
                                    Rollforward Status

   Input database alias                   = zample
   Number of nodes have returned status   = 1

   Node number                            = 0
   Rollforward status                     = not pending
   Next log file to be read               =
   ```

```
   Log files processed                  = S0000000.LOG - S0000000.LOG
   Last committed transaction           = 2004-02-16-20.10.38.000000 UTC

  DB20000I  The ROLLFORWARD command completed successfully.
```

PASSWORDACCESS = prompt environment

In a PROMPT environment, extra information is required, specifically the TSM nodename and password of the machine where the objects were created.

For db2adutl, update the dsm.sys file (called the *dsm.opt* file on Windows-based platforms) and add NODENAME `bar` (because `bar` is the name of the source computer) to the server clause:

```
dps:/home/regress9/sqllib/adsm> db2adutl query db zample nodename bar
owner roecken password *******
```

The following information is returned:

```
Query for database ZAMPLE


Retrieving FULL DATABASE BACKUP information.
    1 Time: 20040216151025  Oldest log: S0000000.LOG  DB Partition Number: 0
    Sessions: 1


Retrieving INCREMENTAL DATABASE BACKUP information.
  No INCREMENTAL DATABASE BACKUP images found for ZAMPLE


Retrieving DELTA DATABASE BACKUP information.
  No DELTA DATABASE BACKUP images found for ZAMPLE


Retrieving TABLESPACE BACKUP information.
  No TABLESPACE BACKUP images found for ZAMPLE


Retrieving INCREMENTAL TABLESPACE BACKUP information.
  No INCREMENTAL TABLESPACE BACKUP images found for ZAMPLE


Retrieving DELTA TABLESPACE BACKUP information.
  No DELTA TABLESPACE BACKUP images found for ZAMPLE


Retrieving LOAD COPY information.
    1 Time: 20040216151213


Retrieving LOG ARCHIVE information.
   Log file: S0000000.LOG, Chain Num: 0, DB Partition Number: 0,
   Taken at: 2004-02-16-15.10.38
```

1. If the database does not exist, create an empty `zample` database. If the `zample` database already exists, this step, and the next two steps that update the database configuration, can be skipped.

   ```
   dps:/home/regress9> db2 create db zample
   ```

2. Update the database configuration parameter *tsm_nodename* for the `zample` database:

   ```
   dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
   ```

3. Update the database configuration parameter *tsm_password* for the `zample` database:

```
dps:/home/regress9> db2 update db cfg for zample using
tsm_password ********
```

4. Restore the `zample` database:

```
dps:/home/regress9> db2 restore db zample use tsm options
"'-fromnode=bar -fromowner=roecken'" without prompting
```

The restore operation completes successfully, but a warning is issued:

```
SQL2540W  Restore is successful, however a warning "2523" was
encountered during Database Restore while processing in No
Interrupt mode.
```

Again, at this point, the rollforward utility cannot find the correct log files:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

The following error message is returned:

```
SQL1268N  Roll-forward recovery stopped due to error "-2112880618"
while retrieving log file "S0000000.LOG" for database "ZAMPLE" on node "0".
```

5. Because the database restore operation replaces the database configuration file, the TSM database configuration values must be set to the correct values. First the *tsm_nodename* configuration parameter must be reset:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_nodename bar
```

6. The *tsm_password* database configuration parameter must be reset:

```
dps:/home/regress9> db2 update db cfg for zample using tsm_password *******
```

7. The *logarchopt1* database configuration parameter must be reset so the rollforward utility can find the correct log files:

```
dps:/home/regress9> db2 update db cfg for zample using logarchopt1
"'-fromnode=bar -fromowner=roecken'"
```

8. The *vendoropt* database configuration parameter must also be reset so that the load recovery file can also be used:

```
dps:/home/regress9> db2 update db cfg for zample using VENDOROPT
"'-fromnode=bar -fromowner=roecken'"
```

9. When the database configuration parameters are set, the database can be rolled forward:

```
dps:/home/regress9> db2 rollforward db zample to end of logs and stop
```

A ROLLFORWARD QUERY STATUS command on the `zample` database shows the following:

```
                                 Rollforward Status

 Input database alias                   = zample
 Number of nodes have returned status   = 1

 Node number                            = 0
 Rollforward status                     = not pending
 Next log file to be read               =
 Log files processed                    = S0000000.LOG - S0000000.LOG
 Last committed transaction             = 2004-02-16-20.10.38.000000 UTC

DB20000I  The ROLLFORWARD command completed successfully.
```

Recovering a dropped table

You might occasionally drop a table that contains data you still need. If this is the case, you should consider making your critical tables recoverable following a drop table operation. You could recover the table data by invoking a database restore operation, followed by a database rollforward operation to a point in time before the table was dropped. This can be time-consuming if the database is large, and your data will be unavailable during recovery. The dropped table recovery feature lets you recover your dropped table data using table space-level restore and rollforward operations. This will be faster than database-level recovery, and your database will remain available to users.

For a dropped table to be recoverable, the table space in which the table resides must have the DROPPED TABLE RECOVERY option turned on. This can be done during table space creation, or by invoking the ALTER TABLESPACE statement. The DROPPED TABLE RECOVERY option is table space-specific and limited to regular table spaces. To determine if a table space is enabled for dropped table recovery, you can query the DROP_RECOVERY column in the SYSCAT.TABLESPACES catalog table.

The dropped table recovery option is on by default when you create a table space. If you do not want to enable a table space for dropped table recovery, you can either explicitly set the DROPPED TABLE RECOVERY option to OFF when you issue the CREATE TABLESPACE statement, or you can use the ALTER TABLESPACE statement to disable dropped table recovery for an existing table space. The dropped table recovery feature may have a performance impact on forward recovery if there are many drop table operations to recover or if the history file is large.

When a DROP TABLE statement is run against a table whose table space is enabled for dropped table recovery, an additional entry (identifying the dropped table) is made in the log files. An entry is also made in the recovery history file, containing information that can be used to re-create the table.

For partitioned tables, dropped table recovery is always on even if the dropped table recovery is turned off for non-partitioned tables in one or more table spaces. Only one dropped table log record is written for a partitioned table. This log record is sufficient to recover all the data partitions of the table.

There are some restrictions on the type of data that is recoverable from a dropped table. It is not possible to recover:

- The DROPPED TABLE RECOVERY option can not be used for temporary table.
- The metadata associated with row types. (The data is recovered, but not the metadata.) The data in the hierarchy table of the typed table will be recovered. This data might contain more information than appeared in the typed table that was dropped.
- XML data. If you attempt to recover a dropped table that contains XML data, the corresponding column data will be empty.

If the table was in reorg pending state when it was dropped, the CREATE TABLE DDL in the history file will not match exactly that of the import file. The import file will be in the format of the table before the first REORG-recommended ALTER was performed, but the CREATE TABLE statement in the history file will match the state of the table including the results of any ALTER TABLE statements."

If the data being recovered is of the GRAPHIC or VARGRAPHIC data type, it might include more than one code page. In order to recover this data, you need to specify the `usegraphiccodepage` file type modifier of the IMPORT or LOAD commands. In this case, using the LOAD command to recover the data will increase the performance of the recovery operation.

Only one dropped table can be recovered at a time. You can recover a dropped table by doing the following:

1. Identify the dropped table by invoking the LIST HISTORY DROPPED TABLE command. The dropped table ID is listed in the Backup ID column.
2. Restore a database- or table space-level backup image taken before the table was dropped.
3. Create an export directory to which files containing the table data are to be written. This directory must either be accessible to all database partitions, or exist on each database partition. Subdirectories under this export directory are created automatically by each database partition. These subdirectories are named NODEnnnn, where nnnn represents the database partition or node number. Data files containing the dropped table data as it existed on each database partition are exported to a lower subdirectory called data. For example,
   ```
   \export_directory\NODE0000\data.
   ```
4. Roll forward to a point in time after the table was dropped, using the RECOVER DROPPED TABLE option on the ROLLFORWARD DATABASE command. Alternatively, roll forward to the end of the logs, so that updates to other tables in the table space or database are not lost.
5. Re-create the table using the CREATE TABLE statement from the recovery history file.
6. Import the table data that was exported during the rollforward operation into the table. If the table was in reorg pending state when the drop took place, the contents of the CREATE TABLE DDL might need to be changed to match the contents of the data file.

Crash recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state. *Crash recovery* is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 17). When a database is in a consistent and usable state, it has attained what is known as a ″point of consistency″.

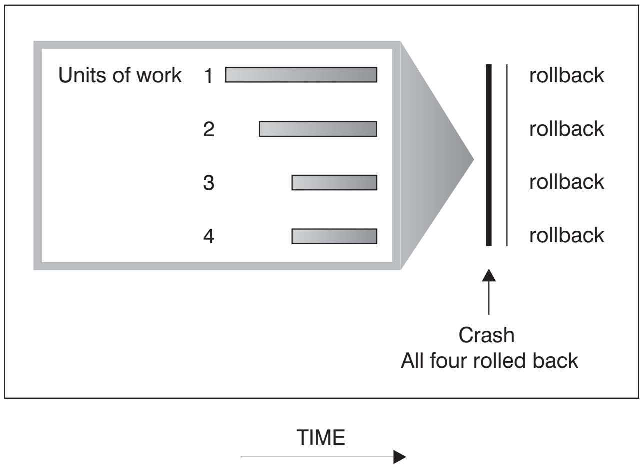


Figure 17. Rolling Back Units of Work (Crash Recovery)

A *transaction failure* results from a severe error or condition that causes the database or the database manager to end abnormally. Partially completed units of work, or UOW that have not been flushed to disk at the time of failure, leave the database in an inconsistent state. Following a transaction failure, the database must be recovered. Conditions that can result in transaction failure include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A hardware failure such as memory corruption, or disk, CPU, or network failure.
- A serious operating system error that causes DB2 to go down
- An application terminating abnormally.

If you want the rollback of incomplete units of work to be done automatically by the database manager, enable the automatic restart (*autorestart*) database configuration parameter by setting it to `ON`. (This is the default value.) If you do not want automatic restart behavior, set the *autorestart* database configuration parameter to `OFF`. As a result, you will need to issue the RESTART DATABASE command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the WRITE RESUME option of the RESTART DATABASE command in order for the crash recovery to continue. The administration notification log records when the database restart operation begins.

If crash recovery is applied to a database that is enabled for forward recovery (that is, the *logarchmeth1* configuration parameter is not set to `OFF`), and an error occurs during crash recovery that is attributable to an individual table space, that table space will be taken offline, and cannot be accessed until it is repaired. Crash recovery continues. At the completion of crash recovery, the other table spaces in the database will be accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections will be permitted.

Recovering damaged table spaces

A damaged table space has one or more containers that cannot be accessed. This is often caused by media problems that are either permanent (for example, a bad disk), or temporary (for example, an offline disk, or an unmounted file system).

If the damaged table space is the system catalog table space, the database cannot be restarted. If the container problems cannot be fixed leaving the original data intact, the only available options are:

- To restore the database
- To restore the catalog table space.

  Note:

  1. Table space restore is only valid for recoverable databases, because the database must be rolled forward.
  2. If you restore the catalog table space, you must perform a rollforward operation to the end of logs.

If the damaged table space is *not* the system catalog table space, DB2 attempts to make as much of the database available as possible.

If the damaged table space is the only temporary table space, you should create a new temporary table space as soon as a connection to the database can be made. Once created, the new temporary table space can be used, and normal database operations requiring a temporary table space can resume. You can, if you want, drop the offline temporary table space. There are special considerations for table reorganization using a system temporary table space:

- If the database or the database manager configuration parameter *indexrec* is set to `RESTART`, all invalid indexes must be rebuilt during database activation; this includes indexes from a reorganization that crashed during the build phase.
- If there are incomplete reorganization requests in a damaged temporary table space, you might have to set the *indexrec* configuration parameter to `ACCESS` to avoid restart failures.

Recovering table spaces in recoverable databases

When crash recovery is necessary, the damaged table space is taken offline and is not accessible. It is placed in roll-forward pending state. If there are no additional problems, a restart operation will succeed in bringing the database online even with the damaged table space. Once online, the damaged table space is unusable, but the rest of the database is usable. To fix the damaged table space and make it usable, follow the procedure below.

To make the damaged table space usable, use one of the procedures that follow:

- Method 1
  1. Fix the damaged containers without losing the original data.
  2. Complete a table space rollforward operation to the end of the logs.

     Note: The rollforward operation will first attempt to bring the table space from offline to normal state.
- Method 2
  1. Fix the damaged containers with or without losing the original data.
  2. Perform a table space restore operation.
  3. Complete a table space rollforward operation to the end of the logs or to a point-in-time.

Recovering table spaces in non-recoverable databases

When crash recovery is needed, but there are damaged table spaces, you can only successfully restart the database if the damaged table spaces are dropped. In a non-recoverable database, the logs necessary to recover the damaged table spaces are not retained. Therefore, the only valid action against such table spaces is to drop them.

To restart a database with damaged table spaces:

1. Invoke an unqualified restart database operation. It will succeed if there are no damaged table spaces. If it fails (SQL0290N), look in the administration notification log file for a complete list of table spaces that are currently damaged.
2. If you are willing to drop all of the damaged table spaces, initiate another restart database operation, listing all of the damaged table spaces under the DROP PENDING TABLESPACES option. If a damaged table space is included in the DROP PENDING TABLESPACES list, the table space is put into drop pending state, and you must drop the table space after the recovery operation is complete.

   The restart operation continues without recovering the specified table spaces. If a damaged table space is *not* included in the DROP PENDING TABLESPACES list, the restart database operation fails with SQL0290N.

Note: Including a table space name in the DROP PENDING TABLESPACES list does not mean that the table space will be in drop pending state. It will be placed in this state only if the table space is found to be damaged during the restart operation.

3. If the restart database operation is successful, invoke the LIST TABLESPACES command to find out which table spaces are in drop pending state.
4. Issue DROP TABLESPACE statements to drop each of the table spaces that are in drop pending state. Once you have done this, you will be able to reclaim the space that the damaged table spaces were using or re-create the table spaces.
5. If you are unwilling to drop and lose the data in the damaged table spaces, you can:
   - Fix the damaged containers (without losing the original data).
   - Reissue the RESTART DATABASE command.
   - Perform a database restore operation.

Reducing the impact of media failure

To reduce the probability of media failure, and to simplify recovery from this type of failure:

- Mirror or duplicate the disks that hold the data and logs for important databases.
- Use a Redundant Array of Independent Disks (RAID) configuration, such as RAID Level 5.
- In a partitioned database environment, set up a rigorous procedure for handling the data and the logs on the catalog partition. Because this database partition is critical for maintaining the database:
  – Ensure that it resides on a reliable disk
  – Duplicate it
  – Make frequent backups
  – Do not put user data on it.

Protecting against disk failure

If you are concerned about the possibility of damaged data or logs due to a disk crash, consider the use of some form of disk fault tolerance. Generally, this is accomplished through the use of a *disk array*, which is a set of disks.

A disk array is sometimes referred to simply as a RAID (Redundant Array of Independent Disks). Disk arrays can also be provided through software at the operating system or application level. The point of distinction between hardware and software disk arrays is how CPU processing of input/output (I/O) requests is handled. For hardware disk arrays, I/O activity is managed by disk controllers; for software disk arrays, this is done by the operating system or an application.

Hardware disk arrays

In a hardware disk array, multiple disks are used and managed by a disk controller, complete with its own CPU. All of the logic required to manage the disks forming this array is contained on the disk controller; therefore, this implementation is operating system-independent.

There are several types of RAID architecture, differing in function and performance, but only RAID level 1 and level 5 are commonly used today.

RAID level 1 is also known as disk mirroring or duplexing. *Disk mirroring* copies data (a complete file) from one disk to a second disk, using a single disk controller. *Disk duplexing* is similar to disk mirroring, except that disks are attached to a second disk controller (like two SCSI adapters). Data protection is good: Either disk can fail, and data is still accessible from the other disk. With disk duplexing, a disk controller can also fail without compromising data protection. Performance is good, but this implementation requires twice the usual number of disks.

RAID level 5 involves data and parity striping by sectors, across all disks. Parity is interleaved with data, rather than being stored on a dedicated drive. Data protection is good: If any disk fails, the data can still be accessed by using information from the other disks, along with the striped parity information. Read performance is good, but write performance is not. A RAID level 5 configuration requires a minimum of three identical disks. The amount of disk space required for overhead varies with the number of disks in the array. In the case of a RAID level 5 configuration with 5 disks, the space overhead is 20 percent.

When using a RAID (but not a RAID level 0) disk array, a failed disk will not prevent you from accessing data on the array. When hot-pluggable or hot-swappable disks are used in the array, a replacement disk can be swapped with the failed disk while the array is in use. With RAID level 5, if two disks fail at the same time, all data is lost (but the probability of simultaneous disk failures is very small).

You might consider using a RAID level 1 hardware disk array or a software disk array for your logs, because this provides recoverability to the point of failure, and offers good write performance, which is important for logs. To do this, use the *mirrorlogpath* configuration parameter to specify a mirror log path on a RAID level 1 file system. In cases where reliability is critical (because time cannot be lost recovering data following a disk failure), and write performance is not so critical, consider using a RAID level 5 hardware disk array. Alternatively, if write performance is critical, and the cost of additional disk space is not significant, consider a RAID level 1 hardware disk array for your data, as well as for your logs.

For detailed information about the available RAID levels, visit the following web site: http://www.acnc.com/04_01_00.html

Software disk arrays

A software disk array accomplishes much the same as does a hardware disk array, but disk traffic is managed by either the operating system, or by an application program running on the server. Like other programs, the software array must compete for CPU and system resources. This is not a good option for a CPU-constrained system, and it should be remembered that overall disk array performance is dependent on the server’s CPU load and capacity.

A typical software disk array provides disk mirroring. Although redundant disks are required, a software disk array is comparatively inexpensive to implement, because costly disk controllers are not required.

CAUTION:
Having the operating system boot drive in the disk array prevents your system from starting if that drive fails. If the drive fails before the disk array is running, the disk array cannot allow access to the drive. A boot drive should be separate from the disk array.

Reducing the impact of transaction failure

To reduce the impact of a transaction failure, try to ensure:

- An uninterrupted power supply for each DB2 server
- Adequate disk space for database logs on all database partitions
- Reliable communication links among the database partition servers in a partitioned database environment
- Synchronization of the system clocks in a partitioned database environment.

Recovering from transaction failures in a partitioned database environment

If a transaction failure occurs in a partitioned database environment, database recovery is usually necessary on both the failed database partition server and any other database partition server that was participating in the transaction:

- Crash recovery occurs on the failed database partition server after the failure condition is corrected.
- *Database partition failure recovery* on the other (still active) database partition servers occurs immediately after the failure has been detected.

In a partitioned database environment, the database partition server on which a transaction is submitted is the coordinator partition, and the first agent that processes the transaction is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT statement for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. During the first phase, the coordinator partition distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

READ-ONLY	No data change occurred at this server
YES	Data change occurred at this server
NO	Because of an error, the server is not prepared to commit

If one of the servers responds with a `NO`, the transaction is rolled back. Otherwise, the coordinator partition begins the second phase.

During the second phase, the coordinator partition writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded with a `YES`. After all the other database partition servers have committed, they send an acknowledgement of the COMMIT to the coordinator partition. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

Transaction failure recovery on an active database partition server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator partition for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.

- If the failed database partition server was the coordinator partition for the application, then agents that are still working for the application on the active servers are interrupted to do failure recovery. The transaction is rolled back locally on each database partition where the transaction is not in prepared state. On those database partitions where the transaction is in prepared state, the transaction becomes in doubt. The coordinator database partition is not aware that the transaction is in doubt on some database partitions because the coordinator database partition is not available.
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator partition, agents working for this application are interrupted. The coordinator partition will either send a ROLLBACK or a DISCONNECT message to the other database partition servers. The transaction will only be in doubt on database partition servers that are still active if the coordinator partition returns SQL0279.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

Transaction failure recovery on the failed database partition server

If the transaction failure causes the database manager to end abnormally, you can issue the db2start command with the RESTART option to restart the database manager once the database partition has been restarted. If you cannot restart the database partition, you can issue db2start to restart the database manager on a different database partition.

If the database manager ends abnormally, database partitions on the server can be left in an inconsistent state. To make them usable, crash recovery can be triggered on a database partition server:

- Explicitly, through the RESTART DATABASE command
- Implicitly, through a CONNECT request when the *autorestart* database configuration parameter has been set to `ON`

Crash recovery reapplies the log records in the active log files to ensure that the effects of all complete transactions are in the database. After the changes have been reapplied, all uncommitted transactions are rolled back locally, *except* for indoubt transactions. There are two types of indoubt transaction in a partitioned database environment:

- On a database partition server that is not the coordinator partition, a transaction is in doubt if it is prepared but not yet committed.
- On the coordinator partition, a transaction is in doubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation occurs when the coordinator agent has not received all the COMMIT acknowledgments from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator partition for an application:

- If the server that restarted is not the coordinator partition for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.
- If the server that restarted *is* the coordinator partition for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery might not be able to resolve all the indoubt transactions. For example, some of the database partition servers might not be available. If the coordinator partition completes crash recovery before other database partitions involved in the transaction, crash recovery will not be able to resolve the indoubt transaction. This is expected because crash recovery is performed by each database partition independently. In this situation, the SQL warning message SQL1061W is returned. Because indoubt transactions hold resources, such as locks and active log space, it is possible to get to a point where no changes can be made to the database because the active log space is being held up by indoubt transactions. For this reason, you should determine whether indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

Note: In a partitioned database server environment, the RESTART database command is run on a per-node basis. In order to ensure that the database is restarted on all nodes, use the following recommended command:

```
db2_all "db2 restart database <database_name>"
```

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the LIST INDOUBT TRANSACTIONS command to query, commit, and roll back the indoubt transaction on the server.

Note: The LIST INDOUBT TRANSACTIONS command is also used in a distributed transaction environment. To distinguish between the two types of indoubt transactions, the *originator* field in the output that is returned by the LIST INDOUBT TRANSACTIONS command displays one of the following:

- DB2 Enterprise Server Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

Identifying the failed database partition server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

SQL0279N
This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

SQL1224N
This SQLCODE is received when the database partition server that failed is the coordinator partition for the transaction.

SQL1229N

This SQLCODE is received when the database partition server that failed is not the coordinator partition for the transaction.

Determining which database partition server failed is a two-step process.

1. Find the partition server that detected the failure by examining the SQLCA. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the db2nodes.cfg file.)
2. Examine the administration notification log on the server found in step one for the node number of the failed server.

Note: If multiple logical nodes are being used on a processor, the failure of one logical node can cause other logical nodes on the same processor to fail.

Recovering from the failure of a database partition server

To recover from the failure of a database partition server, perform the following steps.

1. Correct the problem that caused the failure.
2. Restart the database manager by issuing the db2start command from any database partition server.
3. Restart the database by issuing the RESTART DATABASE command on the failed database partition server or servers.

Recovering indoubt transactions on mainframe or midrange servers

Recovering indoubt transactions on the host when DB2 Connect has the DB2 Syncpoint Manager configured

If your application has accessed a host or System i database server during a transaction, there are some differences in how indoubt transactions are recovered. To access host or System i database servers, DB2 Connect is used. The recovery steps differ if DB2 Connect has the DB2 Syncpoint Manager configured.

The recovery of indoubt transactions at host or System i servers is normally performed automatically by the Transaction Manager (TM) and the DB2 Syncpoint Manager (SPM). An indoubt transaction at a host or System i server does not hold any resources at the local DB2 location, but does hold resources at the host or System i server as long as the transaction is indoubt at that location. If the administrator of the host or System i server determines that a heuristic decision must be made, then the administrator might contact the local DB2 database administrator (for example via telephone) to determine whether to commit or roll back the transaction at the host or System i server. If this occurs, the LIST DRDA® INDOUBT TRANSACTIONS command can be used to determine the state of the transaction at the DB2 Connect instance. The following steps can be used as a guideline for most situations involving an SNA communications environment.

1. Connect to the SPM as shown below:

```
db2 => connect to db2spm

Database Connection Information
Database product       = SPM0500
SQL authorization ID   = CRUS
Local database alias   = DB2SPM
```

2. Issue the LIST DRDA INDOUBT TRANSACTIONS command to display the indoubt transactions known to the SPM. The example below shows one indoubt transaction known to the SPM. The db_name is the local alias for the host or System i server. The partner_lu is the fully qualified luname of the host or System i server. This provides the best identification of the host or System i server, and should be provided by the caller from the host or System i server. The luwid provides a unique identifier for a transaction and is available at all hosts and System i servers. If the transaction in question is displayed, then the uow_status field can be used to determine the outcome of the transaction if the value is C (commit) or R (rollback). If you issue the LIST DRDA INDOUBT TRANSACTIONS command with the WITH PROMPTING parameter, you can commit, roll back, or forget the transaction interactively.

   ```
   db2 => list drda indoubt transactions
   DRDA Indoubt Transactions:
   1.db_name: DBAS3    db_alias: DBAS3    role: AR
     uow_status: C  partner_status: I  partner_lu: USIBMSY.SY12DQA
   corr_tok: USIBMST.STB3327L
   luwid: USIBMST.STB3327.305DFDA5DC00.0001
   xid: 53514C2000000017 00000000544D4442 0000000000305DFD A63055E962000000
        00035F
   ```

3. If an indoubt transaction for the partner_lu and for the luwid is not displayed, or if the LIST DRDA INDOUBT TRANSACTIONS command returns as follows:

   ```
   db2 => list drda indoubt transactions
   SQL1251W  No data returned for heuristic query.
   ```

   then the transaction was rolled back.

There is another unlikely but possible situation that can occur. If an indoubt transaction with the proper luwid for the partner_lu is displayed, but the uow_status is "I", the SPM doesn't know whether the transaction is to be committed or rolled back. In this situation, you should use the WITH PROMPTING parameter to either commit or roll back the transaction on the DB2 Connect workstation. Then allow DB2 Connect to resynchronize with the host or System i server based on the heuristic decision.

Recovering indoubt transactions on the host when DB2 Connect does not use the DB2 Syncpoint Manager

If your application has accessed a host or System i database server during a transaction, there are some differences in how indoubt transactions are recovered. To access host or System i database servers, DB2 Connect is used. The recovery steps differ if DB2 Connect has the DB2 Syncpoint Manager configured.

Use the information in this section when TCP/IP connectivity is used to update DB2 for z/OS in a multisite update from either DB2 Connect Personal Edition or DB2 Connect Enterprise Edition, and the DB2 Syncpoint Manager is not used. The recovery of indoubt transactions in this situation differs from that for indoubt transactions involving the DB2 Syncpoint Manager. When an indoubt transaction occurs in this environment, an alert entry is generated at the client, at the database server, and (or) at the Transaction Manager (TM) database, depending on who detected the problem. The alert entry is placed in the `db2alert.log` file.

The resynchronization of any indoubt transactions occurs automatically as soon as the TM and the participating databases and their connections are all available again. You should allow automatic resynchronization to occur rather than heuristically force a decision at the database server. If, however, you must do this then use the following steps as a guideline.

Note: Because the DB2 Syncpoint Manager is not involved, you cannot use the LIST DRDA INDOUBT TRANSACTIONS command.

1. On the z/OS host, issue the command DISPLAY THREAD TYPE(INDOUBT).

   From this list identify the transaction that you want to heuristically complete. For details about the DISPLAY command, see the *DB2 for z/OS Command Reference*. The LUWID displayed can be matched to the same luwid at the Transaction Manager Database.
2. Issue the RECOVER THREAD(<LUWID>) ACTION(ABORT|COMMIT) command, depending on what you want to do.

   For details about the RECOVER THREAD command, see the *DB2 for z/OS Command Reference*.

Disaster recovery

The term *disaster recovery* is used to describe the activities that need to be done to restore the database in the event of a fire, earthquake, vandalism, or other catastrophic events. A plan for disaster recovery can include one or more of the following:

- A site to be used in the event of an emergency
- A different machine on which to recover the database
- Off-site storage of either database backups, table space backups, or both, as well as archived logs.

If your plan for disaster recovery is to restore the entire database on another machine, it is recommended that you have at least one full database backup and all the archived logs for the database. Although it is possible to rebuild a database if you have a full table space backup of each table space in the database, this method might involve numerous backup images and be more time-consuming than recovery using a full database backup.

You can choose to keep a standby database up to date by applying the logs to it as they are archived. Or, you can choose to keep the database or table space backups and log archives in the standby site, and perform restore and rollforward operations only after a disaster has occurred. (In the latter case, recent backup images are preferable.) In a disaster situation, however, it is generally not possible to recover all of the transactions up to the time of the disaster.

The usefulness of a table space backup for disaster recovery depends on the scope of the failure. Typically, disaster recovery is less complicated and time-consuming if you restore the entire database; therefore, a full database backup should be kept at a standby site. If the disaster is a damaged disk, a table space backup of each table space on that disk can be used to recover. If you have lost access to a container because of a disk failure (or for any other reason), you can restore the container to a different location.

Another way you can protect your data from partial or complete site failures is to implement the DB2 high availability disaster recovery (HADR) feature. Once it is set up, HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby.

You can also protect your data from partial or complete site failures using replication. Replication allows you to copy data on a regular basis to multiple remote databases. DB2 database provides a number of replication tools that allow you to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied.

Storage mirroring, such as Peer-to-Peer Remote Copy (PPRC), can also be used to protect your data. PPRC provides a synchronous copy of a volume or disk to protect against disasters.

DB2 provides you with several options when planning for disaster recovery. Based on your business needs, you might decide to use table space or full database backups as a safeguard against data loss, or you might decide that your environment is better suited to a solution like HADR. Whatever your choice, you should test your recovery procedures in a test environment before implementing them in your production environment.

Version recovery

Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation. You use this recovery method with non-recoverable databases (that is, databases for which you do not have archived logs). You can also use this method with recoverable databases by using the WITHOUT ROLLING FORWARD option on the RESTORE DATABASE command. A database restore operation will restore the entire database using a backup image created earlier. A database backup allows you to restore a database to a state identical to the one at the time that the backup was made. However, every unit of work from the time of the backup to the time of the failure is lost (see Figure 18).

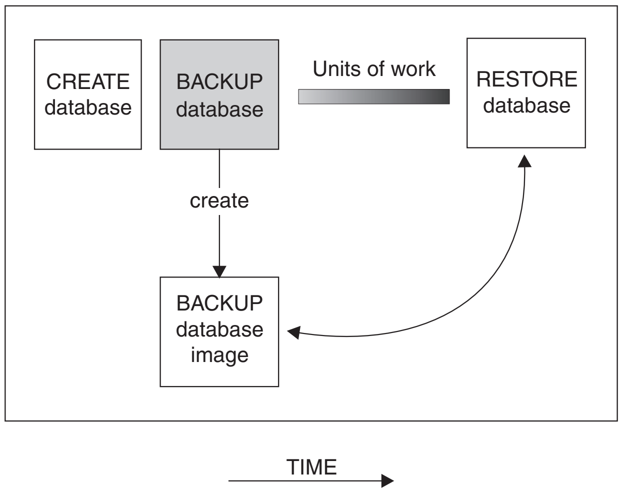


Figure 18. Version Recovery. Shows that units of work from the time of the backup to the time of the failure is lost.

Using the version recovery method, you must schedule and perform full backups of the database on a regular basis.

In a partitioned database environment, the database is located across many database partition servers (or nodes). You must restore all database partitions, and the backup images that you use for the restore database operation must all have been taken at the same time. (Each database partition is backed up and restored separately.) A backup of each database partition taken at the same time is known as a *version backup*.

Rollforward recovery

To use the *rollforward recovery* method, you must have taken a backup of the database and archived the logs (by setting the *logarchmeth1* and *logarchmeth2* configuration parameters to a value other than `OFF`). Restoring the database and specifying the WITHOUT ROLLING FORWARD option is equivalent to using the version recovery method. The database is restored to a state identical to the one at the time that the offline backup image was made. If you restore the database and do *not* specify the WITHOUT ROLLING FORWARD option for the restore database operation, the database will be in rollforward pending state at the end of the restore operation. This allows rollforward recovery to take place.

Note: The WITHOUT ROLLING FORWARD option cannot be used if:

- You are restoring from an online backup image
- You are issuing a table space-level restore

The two types of rollforward recovery to consider are:

- *Database rollforward recovery*. In this type of rollforward recovery, transactions recorded in database logs are applied following the database restore operation (see Figure 19). The database logs record all changes made to the database. This method completes the recovery of the database to its state at a particular point in time, or to its state immediately before the failure (that is, to the end of the active logs.)

  In a partitioned database environment, the database is located across many database partitions, and the ROLLFORWARD DATABASE command must be issued on the database partition where the catalog tables for the database resides (catalog partition). If you are performing point-in-time rollforward recovery, all database partitions must be rolled forward to ensure that all database partitions are at the same level. If you need to restore a single database partition, you can perform rollforward recovery to the end of the logs to bring it up to the same level as the other database partitions in the database. Only recovery to the end of the logs can be used if one database partition is being rolled forward. Point-in-time recovery applies to *all* database partitions.

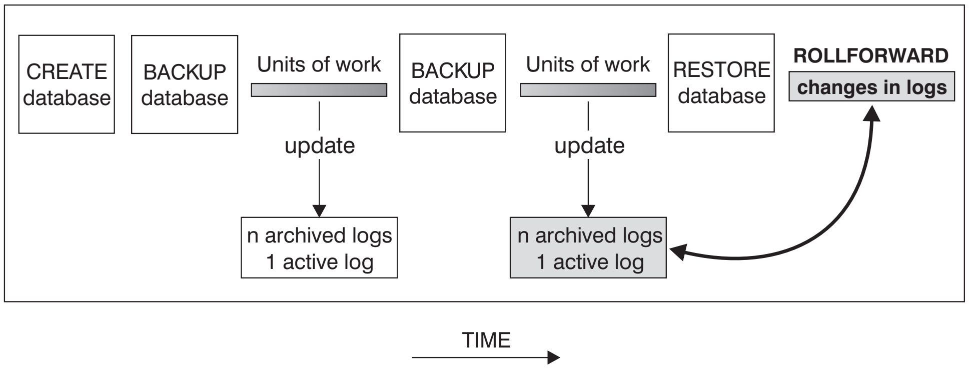


Figure 19. Database Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

- *Table space rollforward recovery*. If the database is enabled for forward recovery, it is also possible to back up, restore, and roll table spaces forward (see Figure 20 on page 234 on page 234). To perform a table space restore and rollforward operation, you need a backup image of either the entire database (that is, all of the table spaces), or one or more individual table spaces. You also need the log records that affect the table spaces that are to be recovered. You can roll forward through the logs to one of two points:

– The end of the logs; or,
– A particular point in time (called *point-in-time* recovery).

Table space rollforward recovery can be used in the following two situations:

- After a table space restore operation, the table space is always in rollforward pending state, and it must be rolled forward. Invoke the ROLLFORWARD DATABASE command to apply the logs against the table spaces to either a point in time, or to the end of the logs.
- If one or more table spaces are in *rollforward pending* state after crash recovery, first correct the table space problem. In some cases, correcting the table space problem does not involve a restore database operation. For example, a power loss could leave the table space in rollforward pending state. A restore database operation is not required in this case. Once the problem with the table space is corrected, you can use the ROLLFORWARD DATABASE command to apply the logs against the table spaces to the end of the logs. If the problem is corrected before crash recovery, crash recovery might be sufficient to take the database to a consistent, usable state.

  Note: If the table space in error contains the system catalog tables, you will not be able to start the database. You must restore the SYSCATSPACE table space, then perform rollforward recovery to the end of the logs.

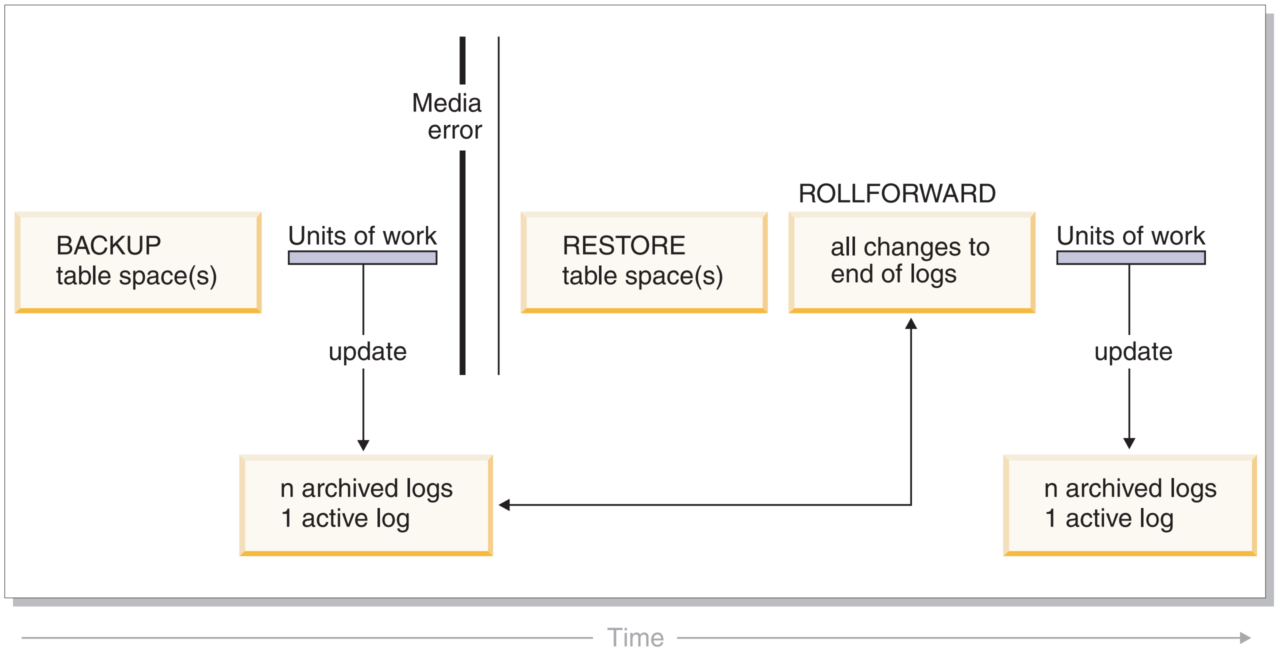


Figure 20. Table Space Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

In a partitioned database environment, if you are rolling a table space forward *to a point in time*, you do not have to supply the list of database partitions on which the table space resides. DB2 submits the rollforward request to all database partitions. This means the table space must be restored on all database partitions on which the table space resides.

In a partitioned database environment, if you are rolling a table space forward *to the end of the logs*, you must supply the list of database partitions if you do *not* want to roll the table space forward on all database partitions. If you want to roll all table spaces (on all database partitions) that are in rollforward pending state forward to the end of the logs, you do not have to supply the list of database partitions. By default, the database rollforward request is sent to all database partitions.

If you are rolling a table space forward that contains any piece of a partitioned table and you are rolling it forward to a point in time, you must also roll all of the other table spaces in which that table resides forward to the same point in time. However, you can roll a single table space containing a piece of a partitioned table forward to the end of logs.

Note: If a partitioned table has any attached, detached, or dropped data partitions, then point-in-time rollforward must also include all table spaces for these data partitions. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSDATAPARTITIONS catalog table.

Incremental backup and recovery

As the size of databases, and particularly warehouses, continues to expand into the terabyte and petabyte range, the time and hardware resources required to back up and recover these databases is also growing substantially. Full database and table space backups are not always the best approach when dealing with large databases, because the storage requirements for multiple copies of such databases are enormous. Consider the following issues:

- When a small percentage of the data in a warehouse changes, it should not be necessary to back up the entire database.
- Appending table spaces to existing databases and then taking only table space backups is risky, because there is no guarantee that nothing outside of the backed up table spaces has changed between table space backups.

To address these issues, DB2 provides incremental backup and recovery. An *incremental backup* is a backup image that contains only pages that have been updated since the previous backup was taken. In addition to updated data and index pages, each incremental backup image also contains all of the initial database metadata (such as database configuration, table space definitions, database history, and so on) that is normally stored in full backup images.

Note:

1. If a table space contains long field or large object data and an incremental backup is taken, all of the long field or large object data will be copied into the backup image if any of the pages in that table space have been modified since the previous backup.
2. If you take an incremental backup of a table space that contains a dirty page (that is, a page that contains data that has been changed but has not yet been written to disk) then all large object data is backed up. Normal data is backed up only if it has changed.

Two types of incremental backup are supported:

- *Incremental*. An incremental backup image is a copy of all database data that has changed since the most recent, successful, full backup operation. This is also known as a cumulative backup image, because a series of incremental backups taken over time will each have the contents of the previous incremental backup image. The predecessor of an incremental backup image is always the most recent successful full backup of the same object.
- *Delta*. A delta, or incremental delta, backup image is a copy of all database data that has changed since the last successful backup (full, incremental, or delta) of the table space in question. This is also known as a differential, or non-cumulative, backup image. The predecessor of a delta backup image is the most recent successful backup containing a copy of each of the table spaces in the delta backup image.

The key difference between incremental and delta backup images is their behavior when successive backups are taken of an object that is continually changing over time. Each successive incremental image contains the entire contents of the previous incremental image, plus any data that has changed, or is new, since the previous full backup was produced. Delta backup images contain only the pages that have changed since the previous image of any type was produced.

Combinations of database and table space incremental backups are permitted, in both online and offline modes of operation. Be careful when planning your backup strategy, because combining database and table space incremental backups implies that the predecessor of a database backup (or a table space backup of multiple table spaces) is not necessarily a single image, but could be a unique set of previous database and table space backups taken at different times.

To restore the database or the table space to a consistent state, the recovery process must begin with a consistent image of the entire object (database or table space) to be restored, and must then apply each of the appropriate incremental backup images in the order described below.

To enable the tracking of database updates, DB2 supports a new database configuration parameter, *trackmod*, which can have one of two accepted values:
- NO. Incremental backup is not permitted with this configuration. Database page updates are not tracked or recorded in any way. This is the default value.
- YES. Incremental backup is permitted with this configuration. When update tracking is enabled, the change becomes effective at the first successful connection to the database. Before an incremental backup can be taken on a particular table space, a full backup of that table space is necessary.

For SMS and DMS table spaces, the granularity of this tracking is at the table space level. In table space level tracking, a flag for each table space indicates whether or not there are pages in that table space that need to be backed up. If no pages in a table space need to be backed up, the backup operation can skip that table space altogether.

Although minimal, the tracking of updates to the database can have an impact on the runtime performance of transactions that update or insert data.

Restoring from incremental backup images

- A restore operation from incremental backup images always consists of the following steps:
  1. Identifying the incremental target image.

Determine the final image to be restored, and request an incremental restore operation from the DB2 restore utility. This image is known as the target image of the incremental restore, because it will be the last image to be restored. The incremental target image is specified using the TAKEN AT parameter in the RESTORE DATABASE command.

2. Restoring the most recent full database or table space image to establish a baseline against which each of the subsequent incremental backup images can be applied.
3. Restoring each of the required full or table space incremental backup images, in the order in which they were produced, on top of the baseline image restored in Step 2.
4. Repeating Step 3 until the target image from Step 1 is read a second time. The target image is accessed twice during a complete incremental restore operation. During the first access, only initial data is read from the image; none of the user data is read. The complete image is read and processed only during the second access.

   The target image of the incremental restore operation must be accessed twice to ensure that the database is initially configured with the correct history, database configuration, and table space definitions for the database that will be created during the restore operation. In cases where a table space has been dropped since the initial full database backup image was taken, the table space data for that image will be read from the backup images but ignored during incremental restore processing.

- There are two ways to restore incremental backup images.
  - For an automatic incremental restore, the RESTORE command is issued only once specifying the target image to be used. DB2 then uses the database history to determine the remaining required backup images and restores them.
  - For a manual incremental restore, the RESTORE command must be issued once for each backup image that needs to be restored (as outlined in the steps above).
- **Automatic Incremental Restore Example**

  To restore a set of incremental backup images using automatic incremental restore, specify the TAKEN AT timestamp option on the RESTORE DATABASE command. Use the time stamp for the last image that you want to restore. For example:

  ```
  db2 restore db sample incremental automatic taken at 20031228152133
  ```

  This will result in the DB2 restore utility performing each of the steps described at the beginning of this section automatically. During the initial phase of processing, the backup image with time stamp 20001228152133 is read, and the restore utility verifies that the database, its history, and the table space definitions exist and are valid.

  During the second phase of processing, the database history is queried to build a chain of backup images required to perform the requested restore operation. If, for some reason this is not possible, and DB2 is unable to build a complete chain of required images, the restore operation terminates, and an error message is returned. In this case, an automatic incremental restore will not be possible, and you will have issue the RESTORE DATABASE command with the INCREMENTAL ABORT option. This will cleanup any remaining resources so that you can proceed with a manual incremental restore.

  Note: It is highly recommended that you not use the FORCE option of the PRUNE HISTORY command. The default operation of this command prevents you from deleting history entries that might be required for recovery from the most recent, full database backup image, but with the FORCE option, it is possible to delete entries that are required for an automatic restore operation.

During the third phase of processing, DB2 will restore each of the remaining backup images in the generated chain. If an error occurs during this phase, you will have to issue the RESTORE DATABASE command with the INCREMENTAL ABORT option to cleanup any remaining resources. You will then have to determine if the error can be resolved before you re-issue the RESTORE command or attempt the manual incremental restore again.

- **Manual Incremental Restore Example**

  To restore a set of incremental backup images, using manual incremental restore, specify the target image using the TAKEN AT timestamp option of the RESTORE DATABASE command and follow the steps outlined above. For example:

  1.
     ```
     db2 restore database sample incremental taken at <ts>
     ```
     where: *<ts>* points to the last incremental backup image (the target image) to be restored.
  2.
     ```
     db2 restore database sample incremental taken at <ts1>
     ```
     where: *<ts1>* points to the initial full database (or table space) image.
  3.
     ```
     db2 restore database sample incremental taken at  <tsX>
     ```
     where: *<tsX>* points to each incremental backup image in creation sequence.
  4. Repeat Step 3, restoring each incremental backup image up to and including image *<ts>*.

  If you are performing a database restore operation, and table space backup images have been produced, the table space images must be restored in the chronological order of their backup time stamps.

  The db2ckrst utility can be used to query the database history and generate a list of backup image time stamps needed for an incremental restore. A simplified restore syntax for a manual incremental restore is also generated. It is recommended that you keep a complete record of backups, and only use this utility as a guide.

Limitations to automatic incremental restore

1. If a table space name has been changed since the backup operation you want to restore from, and you use the new name when you issue a table space level restore operation, the required chain of backup images from the database history will not be generated correctly and an error will occur (SQL2571N).

   Example:
   ```
   db2 backup db sample —> <ts1>
   db2 backup db sample incremental —> <ts2>
   db2 rename tablespace from userspace1 to t1
   db2 restore db sample tablespace ('t1') incremental automatic taken
   at <ts2>

   SQL2571N Automatic incremental restore is unable to proceed.
   Reason code: "3".
   ```
   Suggested workaround: Use manual incremental restore.

2. If you drop a database, the database history will be deleted. If you restore the dropped database, the database history will be restored to its state at the time of the restored backup and all history entries after that time will be lost. If you then attempt to perform an automatic incremental restore that would need to use any of these lost history entries, the RESTORE utility will attempt to restore an incorrect chain of backups and will return an ″out of sequence″ error (SQL2572N).

   Example:
   ```
   db2 backup db sample –> <ts1>
   db2 backup db sample incremental –> <ts2>
   db2 backup db sample incremental delta –> <ts3>
   db2 backup db sample incremental delta –> <ts4>
   db2 drop db sample
   db2 restore db sample incremental automatic taken at <ts2>
   db2 restore db sample incremental automatic taken at <ts4>
   ```
   Suggested workarounds:
   - Use manual incremental restore.
   - Restore the history file first from image <ts4> before issuing an automatic incremental restore.
3. If you restore a backup image from one database into another database and then do an incremental (delta) backup, you can no longer use automatic incremental restore to restore this backup image.

   Example:
   ```
   db2 create db a
   db2 create db b

   db2 update db cfg for a using trackmod on

   db2 backup db a –> ts1
   db2 restore db a taken at ts1 into b

   db2 backup db b incremental –> ts2

   db2 restore db b incremental automatic taken at ts2

   SQL2542N No match for a database image file was found based on the source
   database alias "B" and timestamp "ts1" provided.
   ```
   Suggested workaround:
   - Use manual incremental restore as follows:
     ```
     db2 restore db b incremental taken at ts2
     db2 restore db a incremental taken at ts1 into b
     db2 restore db b incremental taken at ts2
     ```
   - After the manual restore operation into database B, issue a full database backup to start a new incremental chain

Optimizing recovery performance

The following should be considered when thinking about recovery performance:
- You can improve performance for databases that are frequently updated by placing the logs on a separate device. In the case of an online transaction processing (OLTP) environment, often more I/O is needed to write data to the logs than to store a row of data. Placing the logs on a separate device will minimize the disk arm movement that is required to move between a log and the database files.

You should also consider what other files are on the disk. For example, moving the logs to the disk used for system paging in a system that has insufficient real memory will defeat your tuning efforts.

DB2 automatically attempts to minimize the time it takes to complete a backup or restore operation by choosing an optimal value for the number of buffers, the buffer size and the parallelism settings. The values are based on the amount of utility heap memory available, the number of processors available and the database configuration.

- To reduce the amount of time required to complete a restore operation, use multiple source devices.
- If a table contains large amounts of long field and LOB data, restoring it could be very time consuming. If the database is enabled for rollforward recovery, the RESTORE command provides the capability to restore selected table spaces. If the long field and LOB data is critical to your business, restoring these table spaces should be considered against the time required to complete the backup task for these table spaces. By storing long field and LOB data in separate table spaces, the time required to complete the restore operation can be reduced by choosing not to restore the table spaces containing the long field and LOB data. If the LOB data can be reproduced from a separate source, choose the NOT LOGGED option when creating or altering a table to include LOB columns. If you choose not to restore the table spaces that contain long field and LOB data, but you need to restore the table spaces that contain the table, you must roll forward to the end of the logs so that all table spaces that contain table data are consistent.

  Note: If you back up a table space that contains table data without the associated long or LOB fields, you cannot perform point-in-time rollforward recovery on that table space. All the table spaces for a table must be rolled forward simultaneously to the same point in time.
- The following apply for both backup and restore operations:
  - Multiple devices should be used.
  - Do not overload the I/O device controller bandwidth.
- DB2 uses multiple agents to perform both crash recovery and database rollforward recovery. You can expect better performance during these operations, particularly on symmetric multi-processor (SMP) machines; using multiple agents during database recovery takes advantage of the extra CPUs that are available on SMP machines.

  The agent type introduced by parallel recovery is db2agnsc. DB2 chooses the number of agents to be used for database recovery based on the number of CPUs on the machine.

  DB2 distributes log records to these agents so that they can be reapplied concurrently, where appropriate. For example, the processing of log records associated with insert, delete, update, add key, and delete key operations can be parallelized in this way. Because the log records are parallelized at the page level (log records on the same data page are processed by the same agent), performance is enhanced, even if all the work was done on one table.
- When you perform a recover operation, DB2 will automatically choose an optimal value for the number of buffers, the buffer size and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration. Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the UTIL_HEAP_SZ configuration parameter.

Privileges, authorities, and authorization required to use recover

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the recover utility.

Chapter 13. Restore overview

The simplest form of the DB2 RESTORE DATABASE command requires only that you specify the alias name of the database that you want to restore. For example:

```
   db2 restore db sample
```

In this example, because the SAMPLE database exists and will be replaced when the RESTORE DATABASE command is issued, the following message is returned:

```
SQL2539W  Warning!  Restoring to an existing database that is the same as
the backup image database.  The database files will be deleted.
Do you want to continue ? (y/n)
```

If you specify `y`, the restore operation should complete successfully.

A database restore operation requires an exclusive connection: that is, no applications can be running against the database when the operation starts, and the restore utility prevents other applications from accessing the database until the restore operation completes successfully. A table space restore operation, however, can be done online.

A table space is not usable until the restore operation (possibly followed by rollforward recovery) completes successfully.

If you have tables that span more than one table space, you should back up and restore the set of table spaces together.

When doing a partial or subset restore operation, you can use either a table space-level backup image, or a full database-level backup image and choose one or more table spaces from that image. All the log files associated with these table spaces from the time that the backup image was created must exist.

You can restore a database from a backup image taken on a 32–bit level into a 64–bit level, but not vice versa.

The DB2 backup and restore utilities should be used to backup and restore your databases. Moving a fileset from one machine to another is not recommended as this may compromise the integrity of the database.

Using restore

Use the RESTORE DATABASE command to recover a database or table space after a problem such as media or storage failure, or application failure. If you have backed up your database, or individual table spaces, you can re-create them if they have become damaged or corrupted in some way.

When restoring to an *existing* database, you should not be connected to the database that is to be restored: the restore utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the restore operation. When restoring to a *new* database, an instance attachment is required to create the database. When restoring to a *new remote* database, you must first attach to the instance where the new database will reside.

Then, create the new database, specifying the code page and the territory of the server. Restore will overwrite the code page of the destination database with that of the backup image.

The database can be local or remote.

The following restrictions apply to the restore utility:
- You can only use the restore utility if the database has been previously backed up using the DB2 backup utility.
- If users other than the instance owner (on UNIX), or members of the DB2ADMNS or Administrators group (on Windows) attempt to restore a backup image, they will get an error (SQL2061N). If other users need access to the backup image, the file permissions need to be changed after the backup is generated.
- A database restore operation cannot be started while the rollforward process is running.
- You can restore a table space into an existing database only if the table space currently exists, and if it is the same table space; "same" means that the table space was not dropped and then recreated between the backup and the restore operation. The database on disk and in the backup image must be the same.
- You cannot issue a table space-level restore of a table space-level backup to a new database.
- You cannot perform an online table space-level restore operation involving the system catalog tables.
- You cannot restore a backup taken in a single database partition environment into an existing partitioned database environment. Instead you must restore the backup to a single database partition environment and then add database partitions as required.
- When restoring a backup image with one code page into a system with a different codepage, the system code page will be overwritten by the code page of the backup image.
- You cannot use the RESTORE DATABASE command to convert non-automatic storage enabled table spaces to automatic storage enabled table space.

The restore utility can be invoked through the command line processor (CLP), the Restore Database notebook or wizard in the Control Center, or the db2Restore application programming interface (API).

Following is an example of the RESTORE DATABASE command issued through the CLP:

```
db2 restore db sample from D:\DB2Backups taken at 20010320122644
```

To open the Restore wizard:
1. From the Control Center, expand the object tree until you find the database or table space object that you want to restore.
2. Right-click on the object and select Restore from the pop-up menu. The Restore wizard opens.

Detailed information is provided through the contextual help facility within the Control Center.

Restoring from a snapshot backup image

Restoring from a snapshot backup uses the fast copying technology of a storage device to perform the data copying portion of the restore.

Before you begin

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:
- IBM TotalStorage SAN Volume Controller
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS series

You must perform a snapshot backup before you can restore from a snapshot backup. See: "Performing a snapshot backup" on page 201.

Procedure

You can restore from a snapshot backup using the RESTORE DATABASE command with the **USE SNAPSHOT** parameter, or the db2Restore API with the SQLU_SNAPSHOT_MEDIA media type:
RESTORE DATABASE command:

```
db2 restore db sample use snapshot
```

db2Restore API

```
int sampleRestoreFunction( char dbAlias[],
                           char restoredDbAlias[],
                           char user[],
                           char pswd[],
                           char workingPath[] )
{
  db2MediaListStruct mediaListStruct = { 0 };

  rmediaListStruct.locations = &workingPath;
  rmediaListStruct.numLocations = 1;
  rmediaListStruct.locationType = SQLU_SNAPSHOT_MEDIA;


  db2RestoreStruct restoreStruct = { 0 };

  restoreStruct.piSourceDBAlias = dbAlias;
  restoreStruct.piTargetDBAlias = restoredDbAlias;
  restoreStruct.piMediaList = &mediaListStruct;
  restoreStruct.piUsername = user;
  restoreStruct.piPassword = pswd;
  restoreStruct.iCallerAction = DB2RESTORE_STORDEF_NOINTERRUPT;


  struct sqlca sqlca = { 0 };


  db2Restore(db2Version900, &restoreStruct, &sqlca);

  return 0;
}
```

Restoring to an existing database

You can restore any database or table space backup image to an existing database. For a database-level restore, the backup image can differ from the existing database in its alias name, its database name, or its database seed. For a table-space level restore, you can restore a table space into an existing database only if the table space currently exists, and if it is the same table space; ″same″ means that the table space was not dropped and then recreated between the backup and the restore operation. The database on disk and in the backup image must be the same.

A database *seed* is a unique identifier for a database that does not change during the life of the database. The seed is assigned by the database manager when the database is created. DB2 always uses the seed from the backup image.

When restoring to an existing database, the restore utility:

- Deletes table, index, and long field data from the existing database, and replaces it with data from the backup image.
- Replaces table entries for each table space being restored.
- Retains the recovery history file, unless it is damaged or has no entries. If the recovery history file is damaged or contains no entries, the database manager copies the file from the backup image. If you want to replace the recovery history file you can issue the RESTORE command with the REPLACE HISTORY FILE option.
- Retains the authentication type for the existing database.
- Retains the database directories for the existing database. The directories define where the database resides, and how it is cataloged.
- Compares the database seeds. If the seeds are different:
  - Deletes the logs associated with the existing database.
  - Copies the database configuration file from the backup image.
  - Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command.

  If the database seeds are the same:
  - Deletes the logs if the image is for a non-recoverable database.
  - Retains the current database configuration file.
  - Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command; otherwise, copies the current log path to the database configuration file. Validates the log path: If the path cannot be used by the database, changes the database configuration to use the default log path.

Restoring to a new database

You can create a new database and then restore a full database backup image to it. If you do not create a new database, the restore utility will create one.

When restoring to a new database, the restore utility:

- Creates a new database, using the database alias name that was specified through the target database alias parameter. (If a target database alias was not specified, the restore utility creates the database with an alias that is the same as that specified through the source database alias parameter.)
- Restores the database configuration file from the backup image.

- Sets NEWLOGPATH to the value of the *logpath* database configuration parameter if NEWLOGPATH was specified on the RESTORE DATABASE command. Validates the log path: If the path cannot be used by the database, changes the database configuration to use the default log path.
- Restores the authentication type from the backup image.
- Restores the comments from the database directories in the backup image.
- Restores the recovery history file for the database.
- Overwrites the code page of the database with the codepage of the backup image.

Using incremental restore in a test and production environment

Once a production database is enabled for incremental backup and recovery, you can use an incremental or delta backup image to create or refresh a test database. You can do this by using either manual or automatic incremental restore. To restore the backup image from the production database to the test database, use the INTO *target-database-alias* option on the RESTORE DATABASE command. For example, in a production database with the following backup images:

```
backup db prod
Backup successful. The timestamp for this backup image is : <ts1>

backup db prod incremental
Backup successful. The timestamp for this backup image is : <ts2>
```

an example of a manual incremental restore would be:

```
restore db prod incremental taken at <ts2> into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at <ts1> into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.

restore db prod incremental taken at <ts2> into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

If the database TEST already exists, the restore operation will overwrite any data that is already there. If the database TEST does not exist, the restore utility will create it and then populate it with the data from the backup images.

Since automatic incremental restore operations are dependent on the database history, the restore steps change slightly based on whether or not the test database exists. To perform an automatic incremental restore to the database TEST, its history must contain the backup image history for database PROD. The database history for the backup image will replace any database history that already exists for database TEST if:

- the database TEST does not exist when the RESTORE DATABASE command is issued, or
- the database TEST exists when the RESTORE DATABASE command is issued, and the database TEST history contains no records.

The following example shows an automatic incremental restore to database TEST which does not exist:

```
restore db prod incremental automatic taken at <ts2> into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

The restore utility will create the TEST database and populate it.

If the database TEST does exist and the database history is not empty, you must drop the database before the automatic incremental restore operation as follows:

```
drop db test
DB20000I  The DROP DATABASE command completed successfully.

restore db prod incremental automatic taken at <ts2> into test without
prompting
DB20000I  The RESTORE DATABASE command completed successfully.
```

If you do not want to drop the database, you can issue the PRUNE HISTORY command using a timestamp far into the future and the WITH FORCE OPTION parameter before issuing the RESTORE DATABASE command:

```
connect to test
Database Connection Information

Database server           = <server id>
SQL authorization ID      = <id>
Local database alias      = TEST

prune history 9999 with force option
DB20000I  The PRUNE command completed successfully.

connect reset
DB20000I  The SQL command completed successfully.
restore db prod incremental automatic taken at <ts2> into test without
prompting
SQL2540W  Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

In this case, the RESTORE DATABASE COMMAND will act in the same manner as when the database TEST did not exist.

If the database TEST does exist and the database history is empty, you do not have to drop the database TEST before the automatic incremental restore operation:

```
restore db prod incremental automatic taken at <ts2> into test without
prompting
SQL2540W  Restore is successful, however a warning "2539" was
encountered during Database Restore while processing in No
Interrupt mode.
```

You can continue taking incremental or delta backups of the test database without first taking a full database backup. However, if you ever need to restore one of the incremental or delta images you will have to perform a manual incremental restore. This is because automatic incremental restore operations require that each of the backup images restored during an automatic incremental restore be created from the same database alias.

If you make a full database backup of the test database after you complete the restore operation using the production backup image, you can take incremental or delta backups and can restore them using either manual or automatic mode.

Performing a redirected restore operation

A redirected restore operation is performed when one of the following situations occur:

- You want to restore a backup image to a target machine that is different than the source machine
- You want to restore your table space containers into a different physical location
- Your restore operation has failed because one or more containers is inaccessible

Note: A redirected restore cannot be used to move data from one operating system to another.

During a redirected restore operation, directory and file containers are automatically created if they do not already exist. The database manager does not automatically create device containers.

DB2 only supports adding, changing or removing table space containers of a DMS table space. For an SMS table space, redirected restore is the only method to modify the table space container configuration.

You can redefine table space containers by invoking the RESTORE DATABASE command and specifying the REDIRECT parameter, or by using the Restore Database wizard in the Control Center. The process for invoking a redirected restore of an incremental backup image is similar to the process for invoking a redirected restore of a non-incremental backup image. Issue the RESTORE DATABASE command with the REDIRECT option and specify the backup image that should be used for the incrementally restore of the database. Alternatively, you can generate a redirected restore script from a backup image, then you can modify the script as required. See "Performing a redirected restore using an automatically generated script" on page 252.

Container redirection provides considerable flexibility for managing table space containers. For example, even though adding containers to SMS table spaces is not supported, you could accomplish this by specifying an additional container when invoking a redirected restore operation.

Example

A redirected restore operation consists of a two-step database restore process with an intervening table space container definition step:

1. Issue the RESTORE DATABASE command with the REDIRECT option.
2. Use the SET TABLESPACE CONTAINERS command to define table space containers for the restored database (specifies the table space locations on the target system)
3. Issue the RESTORE DATABASE command again, this time specifying the CONTINUE option.

The following example shows how to perform a redirected restore on database SAMPLE:

```
db2 restore db sample redirect without prompting
SQL1277W A redirected restore operation is being performed.
Table space configuration can now be viewed and table spaces that do not
use automatic storage can have their containers reconfigured.

DB20000I The RESTORE DATABASE command completed successfully.
```

```
db2 set tablespace containers for 2 using (path 'userspace1.0', path
'userspace1.1')
DB20000I The SET TABLESPACE CONTAINERS command completed successfully.

db2 restore db sample continue
DB20000I The RESTORE DATABASE command completed successfully.
```

Redefine table space containers by restoring a database using an automatically generated script

When you restore a database, the restore utility assumes that the physical container layout will be identical to that of the database when it was backed up. If you need to change the location or size of any of the physical containers, you must issue the RESTORE DATABASE command with the REDIRECT option. Using this option requires that you specify the locations of physical containers stored in the backup image and provide the complete set of containers for each non-automatic table space that will be altered. You can capture the container information at the time of the backup, but this can be cumbersome.

To make it easier to perform a redirected restore, the restore utility allows you to generate a redirected restore script from an existing backup image by issuing the RESTORE DATABASE command with the REDIRECT option and the GENERATE SCRIPT option. The restore utility examines the backup image, extracts container information from the backup image, and generates a CLP script that includes all of the detailed container information. You can then modify any of the paths or container sizes in the script, then run the CLP script to recreate the database with the new set of containers. The script you generate can be used to restore a database even if you only have a backup image and you do not know the layout of the containers. The script is created on the client. Using the script as your basis, you can decide where the restored database will require space for log files and containers and you can change the log file and container paths accordingly.

The generated script consists of four sections:

Initialization

The first section sets command options and specifies the database partitions on which the command will run. The following is an example of the first section:

```
UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM 0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
```

where

- `S ON` specifies that execution of the command should stop if a command error occurs
- `Z ON SAMPLE_NODE0000.out` specifies that output should be directed to a file named `<dbalias>_NODE<dbpartitionnum>.out`
- `V ON` specifies that the current command should be printed to standard output.

  When running the script on a partitioned database environment, it is important to specify the database partition on which the script commands will run.

RESTORE command with the REDIRECT option

The second section starts the RESTORE command and uses the REDIRECT option. This section can use all of the RESTORE command options, except any options that cannot be used in conjunction with the REDIRECT option. The following is an example of the second section:

```
RESTORE DATABASE SAMPLE
-- USER '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050906194027
-- DBPATH ON '<target-directory>'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
```

Table space definitions

This section contains table space definitions for each table space in the backup image or specified on the command line. There is a section for each table space, consisting of a comment block that contains information about the name, type and size of the table space. The information is provided in the same format as a table space snapshot. You can use the information provided to determine the required size for the table space. In cases where you are viewing output of a table space created using automatic storage, you will not see a SET TABLESPACE CONTAINERS clause. The following is an example of the table space definition section:

```
-- *********************************************************************
-- ** Tablespace name                            = SYSCATSPACE
-- **   Tablespace ID                            = 0
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Total number of pages                    = 5572
-- *********************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- *********************************************************************
-- ** Tablespace name                            = TEMPSPACE1
-- **   Tablespace ID                            = 1
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = System Temporary data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Total number of pages                    = 0
-- *********************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- *********************************************************************
-- ** Tablespace name                            = DMS
-- **   Tablespace ID                            = 2
-- **   Tablespace Type                          = Database managed space
-- **   Tablespace Content Type                 = Any data
-- **   Tablespace Page size (bytes)            = 4096
-- **   Tablespace Extent size (pages)          = 32
-- **   Using automatic storage                 = No
-- **   Auto-resize enabled                     = No
-- **   Total number of pages                   = 2000
-- **   Number of usable pages                  = 1960
-- **   High water mark (pages)                 = 96
-- ******************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                        1000
, FILE   '/tmp/dms2'                                        1000
);
```

RESTORE command with the CONTINUE option

The final section issues the RESTORE command with the CONTINUE option, to complete the redirected restore. The following is an example of the final section:

```
RESTORE DATABASE SAMPLE CONTINUE;
```

Performing a redirected restore using an automatically generated script

When you perform a redirected restore operation, you need to specify the locations of physical containers stored in the backup image and provide the complete set of containers for each table space that will be altered. Use the following procedure to generate a redirected restore script based on an existing backup image, modify the generated script, then run the script to perform the redirected restore.

You can perform a redirected restore only if the database has been previously backed up using the DB2 backup utility.

- If the database exists, you must be able to connect to it in order to generate the script. Therefore, if the database requires an upgrade or crash recovery, this must be done before you attempt to generate a redirected restore script.
- If you are working in a partitioned database environment, and the target database does not exist, you cannot run the command to generate the redirected restore script concurrently on all database partitions. Instead, the command to generate the redirected restore script must be run one database partition at a time, starting from the catalog partition.

  Alternatively, you can first create a dummy database with the same name as your target database. After the dummy database has been created, you can then generate the redirected restore script concurrently on all database partitions.
- Even if you specify the REPLACE EXISTING option when you issue the RESTORE command to generate the script, the REPLACE EXISTING option will appear in the script commented out.
- For security reasons, your password will not appear in the generated script. You need to fill in the password manually.
- You cannot generate a script for redirected restore using the Restore Wizard in the Control Center.

To perform a redirected restore using a script:

1. Use the restore utility to generate a redirected restore script. The restore utility can be invoked through the command line processor (CLP) or the db2Restore

application programming interface (API). The following is an example of the RESTORE DATABASE command with the REDIRECT option and the GENERATE SCRIPT option:

```
db2 restore db test from /home/jseifert/backups taken at 20050304090733
      redirect generate script test_node0000.clp
```

This creates a redirected restore script on the client called `test_node0000.clp`.

2. Open the redirected restore script in a text editor to make any modifications that are required. You can modify:
   - Restore options
   - Automatic storage paths
   - Container layout and paths
3. Run the modified redirected restore script. For example:

   ```
   db2 -tvf test_node0000.clp
   ```

Database rebuild

Rebuilding a database is the process of restoring a database or a subset of its table spaces using a set of restore operations. The functionality provided with database rebuild makes DB2 more robust and versatile, and provides you with a more complete recovery solution.

The ability to rebuild a database from table space backup images means that you no longer have to take as many full database backups. As databases grow in size, opportunities for taking a full database backup are becoming limited. With table space backup as an alternative, you no longer need to take full database backups as frequently. Instead, you can take more frequent table space backups and plan to use them, along with log files, in case of a disaster.

In a recovery situation, if you need to bring a subset of table spaces online faster than others, you can use rebuild to accomplish this. The ability to bring only a subset of table spaces online is especially useful in a test and production environment.

Rebuilding a database involves a series of potentially many restore operations. A rebuild operation can use a database image, or table space images, or both. It can use full backups, or incremental backups, or both. The initial restore operation restores the target image, which defines the structure of the database that can be restored (such as the table space set and the database configuration). For recoverable databases, rebuilding allows you to build a database that is connectable and that contains the subset of table spaces that you need to have online, while keeping table spaces that can be recovered at a later time offline.

The method you use to rebuild your database depends on whether it is recoverable or non-recoverable.

- If the database is recoverable, use one of the following methods:
  – Using a full or incremental database or table space backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image only using the REBUILD option. You can then roll your database forward to a point in time.
  – Using a full or incremental database or table space backup image as your target, rebuild your database by specifying the set of table spaces defined in the database at the time of the target image to be restored using the REBUILD option. SYSCATSPACE must be part of this set. This operation will restore

    those table spaces specified that are defined in the target image and then use the recovery history file to find and restore any other required backup images for the remaining table spaces not in the target image automatically. Once the restores are complete, roll your database forward to a point in time.
- If the database is non-recoverable:
  - Using a full or incremental database backup image as your target, rebuild your database by restoring SYSCATSPACE and any other table spaces from the target image using the appropriate REBUILD syntax. When the restore completes you can connect to the database.

Specifying the target image

To perform a rebuild of a database, you start by issuing the RESTORE command, specifying the most recent backup image that you use as the target of the restore operation. This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. The rebuild target image is specified using the TAKEN AT parameter in the RESTORE DATABASE command. The target image can be any type of backup (full, table space, incremental, online or offline). The table spaces defined in the database at the time the target image was created will be the table spaces available to rebuild the database.

You must specify the table spaces you want restored using one of the following methods:
- Specify that you want all table spaces defined in the database to be restored and provide an exception list if there are table spaces you want to exclude
- Specify that you want all table spaces that have user data in the target image to be restored and provide an exception list if there are table spaces you want to exclude
- Specify the list of table spaces defined in the database that you want to restore

Once you know the table spaces you want the rebuilt database to contain, issue the RESTORE command with the appropriate REBUILD option and specify the target image to be used.

Rebuild phase

After you issue the RESTORE command with the appropriate REBUILD option and the target image has been successfully restored, the database is considered to be in the rebuild phase. After the target image is restored, any additional table space restores that occur will restore data into existing table spaces, as defined in the rebuilt database. These table spaces will then be rolled forward with the database at the completion of the rebuild operation.

If you issue the RESTORE command with the appropriate REBUILD option and the database does not exist, a new database is created based on the attributes in the target image. If the database does exist, you will receive a warning message notifying you that the rebuild phase is starting. You will be asked if you want to continue the rebuild operation or not.

The rebuild operation restores all initial metadata from the target image. This includes all data that belongs to the database and does not belong to the table space data or the log files. Examples of initial metadata are:

- Table spaces definitions
- The history file, which is a database file that records administrative operations

The rebuild operation also restores the database configuration. The target image sets the log chain that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

If a database already exists on disk and you want the history file to come from the target image, then you should specify the REPLACE HISTORY FILE option. The history file on disk at this time is used by the automatic logic to find the remaining images needed to rebuild the database.

Once the target image is restored:
- if the database is recoverable, the database is put into roll-forward pending state and all table spaces that you restore are also put into roll-forward pending state. Any table spaces defined in the database but not restored are put in restore pending state.
- If the database is not recoverable, then the database and the table spaces restored will go into normal state. Any table spaces not restored are put in drop pending state, as they can no longer be recovered. For this type of database, the rebuild phase is complete.

For recoverable databases, the rebuild phase ends when the first ROLLFORWARD DATABASE command is issued and the rollforward utility begins processing log records. If a rollforward operation fails after starting to process log records and a restore operation is issued next, the restore is not considered to be part of the rebuild phase. Such restores should be considered as normal table space restores that are not part of the rebuild phase.

Automatic processing

After the target image is restored, the restore utility determines if there are remaining table spaces that need to be restored. If there are, they are restored using the same connection that was used for running the RESTORE DATABASE command with the REBUILD option. The utility uses the history file on disk to find the most recent backup images taken prior to the target image that contains each of the remaining table spaces that needs to be restored. The restore utility uses the backup image location data stored in the history file to restore each of these images automatically. These subsequent restores, which are table space level restores, can be performed only offline. If the image selected does not belong on the current log chain, an error is returned. Each table space that is restored from that image is placed in roll-forward pending state.

The restore utility tries to restore all required table spaces automatically. In some cases, it will not be able to restore some table spaces due to problems with the history file, or an error will occur restoring one of the required images. In such a case, you can either finish the rebuild manually or correct the problem and re-issue the rebuild.

If automatic rebuilding cannot complete successfully, the restore utility writes to the diagnostics log (db2diag log file) any information it gathered for the remaining restore steps. You can use this information to complete the rebuild manually.

If a database is being rebuilt, only containers belonging to table spaces that are part of the rebuild process will be acquired.

If any containers need to be redefined through redirected restore, you will need to set the new path and size of the new container for the remaining restores and the subsequent rollforward operation.

If the data for a table space restored from one of these remaining images cannot fit into the new container definitions, the table space is put into restore pending state and a warning message is returned at the end of the restore. You can find additional information about the problem in the diagnostic log.

Completing the rebuild phase

Once all the intended table spaces have been restored you have two options based on the configuration of the database. If the database is non-recoverable, the database will be connectable and any table spaces restored will be online. Any table spaces that are in drop pending state can no longer be recovered and should be dropped if future backups will be performed on the database.

If the database is recoverable, you can issue the rollforward command to bring the table spaces that were restored online. If SYSCATSPACE has not been restored, the rollforward will fail and this table space will have to be restored before the rollforward operation can begin. This means that during the rebuild phase, SYSCATSPACE must be restored.

Note: In a partitioned database environment, SYSCATSPACE does not exist on non-catalog partitions so it cannot be rebuilt there. However, on the catalog partition, SYSCATSPACE must be one of the table spaces that is rebuilt, or the rollforward operation will not succeed.

Rolling the database forward brings the database out of roll-forward pending state and rolls any table spaces in roll-forward pending state forward. The rollforward utility will not operate on any table space in restore pending state.

The stop time for the rollforward operation must be a time that is later than the end time of the most recent backup image restored during the rebuild phase. An error will occur if any other time is given. If the rollforward operation is not able to reach the backup time of the oldest image that was restored, the rollforward utility will not be able to bring the database up to a consistent point, and the rollforward fails.

You must have all log files for the time frame between the earliest and most recent backup images available for the rollforward utility to use. The logs required are those logs which follow the log chain from the earliest backup image to the target backup image, as defined by the truncation array in the target image, otherwise the rollforward operation will fail. If any backup images more recent than the target image were restored during the rebuild phase, then the additional logs from the target image to the most recent backup image restored are required. If the logs are not made available, the rollforward operation will put those table spaces that were not reached by the logs into restore pending state. You can issue the LIST HISTORY command to show the restore rebuild entry with the log range that will be required by roll forward.

The correct log files must be available. If you rely on the rollforward utility to retrieve the logs, you must ensure that the DB2 Log Manager is configured to indicate the location from which log files can be retrieved. If the log path or archive path has changed, you need to use the OVERFLOW LOG PATH option of the ROLLFORWARD DATABASE command.

Use the AND STOP option of the ROLLFORWARD DATABASE command to make the database available when the rollforward command successfully completes. At this point, the database is no longer in roll-forward pending state. If the rollforward operation begins, but an error occurs before it successfully completes, the rollforward operation stops at the point of the failure and an error is returned. The database remains in roll-forward pending state. You must take steps to correct the problem (for example, fix the log file) and then issue another rollforward operation to continue processing.

If the error cannot be fixed, you will be able to bring the database up at the point of the failure by issuing the ROLLFORWARD STOP command. Any log data beyond that point in the logs will no longer be available once the STOP option is used. The database comes up at that point and any table spaces that have been recovered are online. Table spaces that have not yet been recovered are in restore pending state. The database is in the normal state.

You will have to decide what is the best way to recover the remaining table spaces in restore pending state. This could be by doing a new restore and roll forward of a table space or by re-issuing the whole rebuild operation again. This will depend on the type of problems encountered. In the situation where SYSCATSPACE is one of the table spaces in restore pending state, the database will not be connectable.

Rebuild and table space containers

During a rebuild, only those table spaces that are part of the rebuild process will have their containers acquired. The containers belonging to each table space will be acquired at the time the table space user data is restored out of an image.

When the target image is restored, each table space known to the database at the time of the backup will have its definitions only restored. This means the database created by the rebuild will have knowledge of the same table spaces it did at backup time. For those table spaces that should also have their user data restored from the target image, their containers will also be acquired at this time.

Any remaining table spaces that are restored through intermediate table space restores will have their containers acquired at the time the image is restored that contains the table space data.

Rebuild with redirected restore

In the case of redirected restore, all table space containers must be defined during the restore of the target image. If you specify the REDIRECT option, control will be given back to you to redefine your table space containers. If you have redefined table space containers using the SET TABLESPACE CONTAINERS command then those new containers will be acquired at that time. Any table space containers that you have not redefined will be acquired as normal, at the time the table space user data is restored out of an image.

If the data for a table space that is restored cannot fit into the new container definitions, the table space will be put into restore-pending state and a warning (SQL2563W) will be returned to the you at the end of the restore. There will be a message in the DB2 diagnostics log detailing the problem.

Rebuild and temporary table spaces

In general, a DB2 backup image is made up of the following components:

- Initial database metadata, such as the table space definitions, database configuration file, and history file.
- Data for non-temporary table spaces specified to the BACKUP utility
- Final database metadata such as the log file header
- Log files (if the INCLUDE LOGS option was specified)

In every backup image, whether it is a database or table space backup, a full or incremental (delta) backup, these core components can always be found.

A database backup image will contain all of the above components, as well as data for every table space defined in the database at the time of the backup.

A table space backup image will always include the database metadata listed above, but it will only contain data for those table spaces that are specified to the backup utility.

Temporary table spaces are treated differently than non-temporary table spaces. Temporary table space data is never backed up, but their existence is important to the framework of the database. Although temporary table space data is never backed up, the temporary table spaces are considered part of the database, so they are specially marked in the metadata that is stored with a backup image. This makes it look like they are in the backup image. In addition, the table space definitions hold information about the existence of any temporary table spaces.

Although no backup image ever contains data for a temporary table space, during a database rebuild operation when the target image is restored (regardless the type of image), temporary table spaces are also restored, only in the sense that their containers are acquired and allocated. The acquisition and allocation of containers is done automatically as part of the rebuild processing. As a result, when rebuilding a database, you cannot exclude temporary table spaces.

Choosing a target image for database rebuild

The rebuild target image should be the most recent backup image that you want to use as the starting point of your restore operation. This image is known as the target image of the rebuild operation, because it defines the structure of the database to be restored, including the table spaces that can be restored, the database configuration, and the log sequence. It can be any type of backup (full, table space, incremental, online or offline).

The target image sets the log sequence (or log chain) that determines what images can be used for the remaining restores during the rebuild phase. Only images on the same log chain can be used.

The following examples illustrate how to choose the image you should use as the target image for a rebuild operation.

Suppose there is a database called SAMPLE that has the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

Figure 21 shows that the following database-level backups and table space-level backups that have been taken, in chronological order:

1. Full database backup DB1
2. Full table space backup TS1
3. Full table space backup TS2
4. Full table space backup TS3
5. Database restore and roll forward to a point between TS1 and TS2
6. Full table space backup TS4
7. Full table space backup TS5

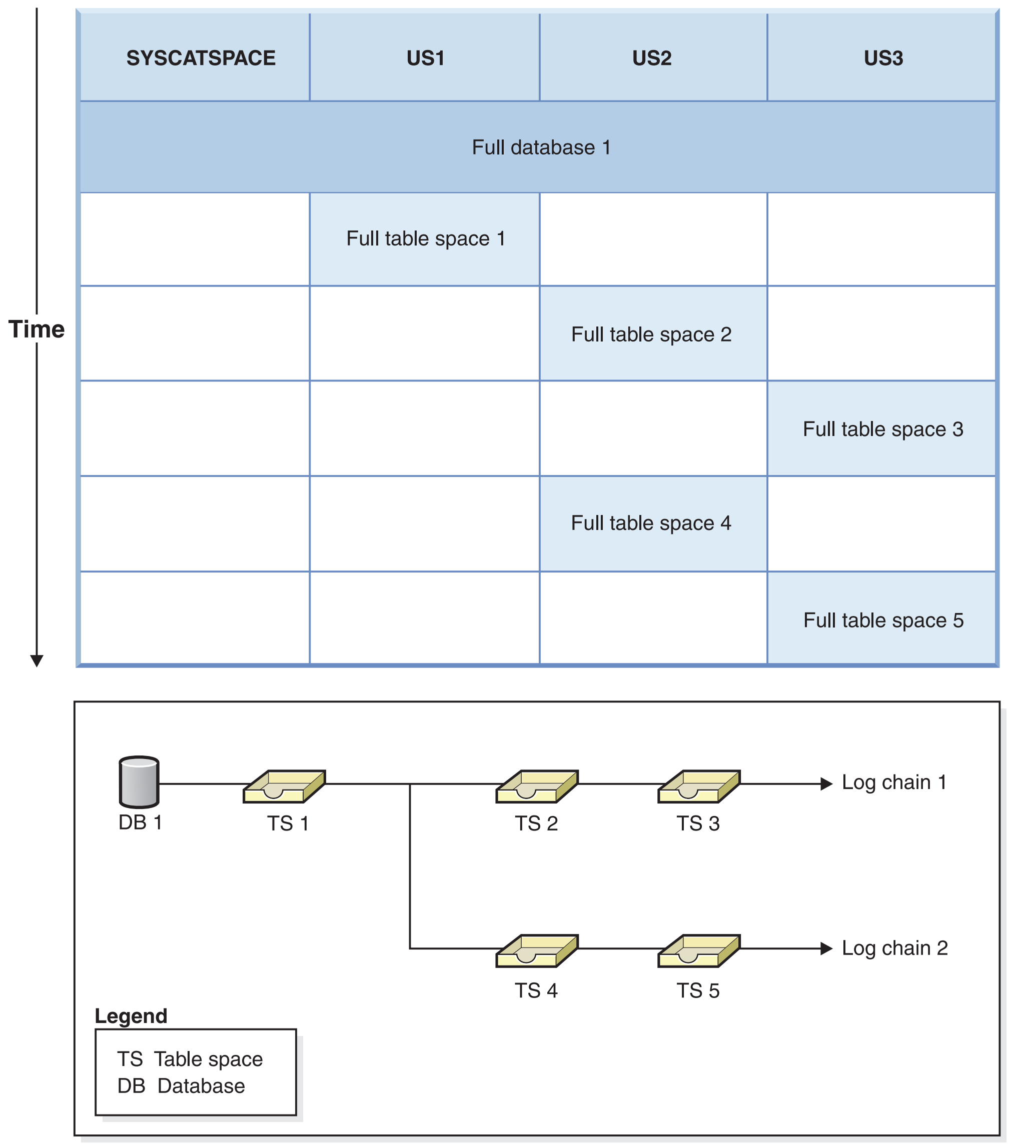


Figure 21. Database and table space-level backups of database SAMPLE

Example 1

The following example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the current point of time. First you need to choose the table spaces you want to rebuild. Since your goal is to rebuild the database to the current point of time you need to select the most recent backup image as your target image. The most recent backup image is image TS5, which is on log chain 2:

```
db2 restore db sample rebuild with all tablespaces in database taken at
   TS5 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
```

This restores backup images TS5, TS4, TS1 and DB1 automatically, then rolls the database forward to the end of log chain 2.

Note: All logs belonging to log chain 2 must be accessible for the rollforward operations to complete.

Example 2

This second example demonstrates the CLP commands you need to issue to rebuild database SAMPLE to the end of log chain 1. The target image you select should be the most recent backup image on log chain 1, which is TS3:

```
db2 restore db sample rebuild with all tablespaces in database
   taken at TS3 without prompting
db2 rollforward db sample to end of logs
db2 rollforward db sample stop
```

This restores backup images TS3, TS2, TS1 and DB1 automatically, then rolls the database forward to the end of log chain 1.

Note: All logs belonging to log chain 1 must be accessible for the rollforward operations to complete.

Choosing the wrong target image for rebuild

Suppose there is a database called SAMPLE2 that has the following table spaces in it:
- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Figure 22 on page 261 shows the backup log chain for SAMPLE2, which consists of the following backups:
1. BK1 is a full database backup, which includes all table spaces
2. BK2 is a full table space backup of USERSP1
3. BK3 is a full table space backup of USERSP2

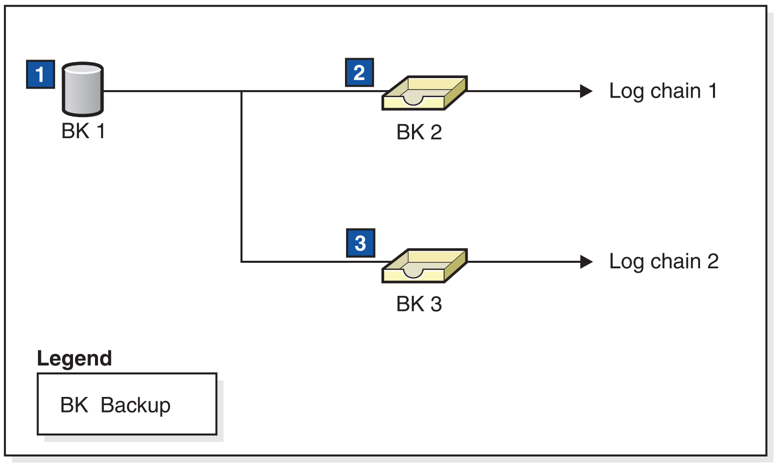


Figure 22. Backup log chain for database SAMPLE2

The following example demonstrates the CLP command you need to issue to rebuild the database from BK3 using table spaces SYSCATSPACE and USERSP2:

```
db2 restore db sample2 rebuild with tablespace (SYSCATSPACE,
   USERSP2) taken at BK3 without prompting
```

Now suppose that after this restore completes, you decide that you also want to restore USERSP1, so, you issue the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK2
```

This restore fails and provides a message that says BK2 is from the wrong log chain (SQL2154N). As you can see in Figure 22, the only image that can be used to restore USERSP1 is BK1. Therefore, you need to type the following command:

```
db2 restore db sample2 tablespace (USERSP1) taken at BK1
```

This succeeds so that database can be rolled forward accordingly.

Rebuilding selected table spaces

Rebuilding a database allows you to build a database that contains a subset of the table spaces that make up the original database. Rebuilding only a subset of table spaces within a database can be useful in the following situations:

- In a test and development environment in which you want to work on only a subset of table spaces.
- In a recovery situation in which you need to bring table spaces that are more critical online faster than others, you can first restore a subset of table spaces then restore other table spaces at a later time.

To rebuild a database that contains a subset of the table spaces that make up the original database, consider the following example.

In this example, there is a database named SAMPLE that has the following table spaces:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:
- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

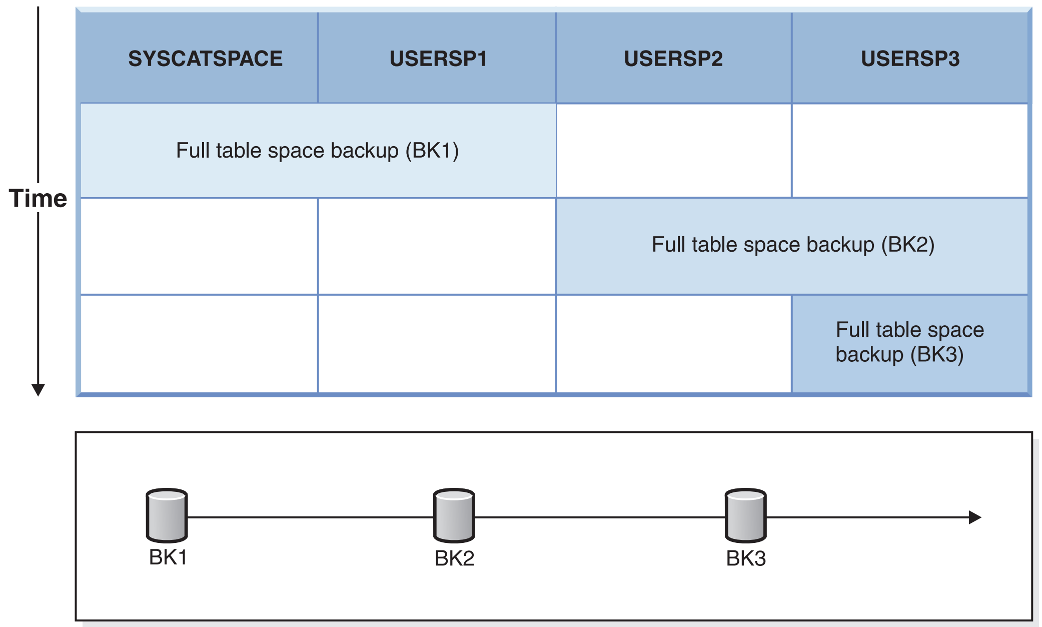

Time
SYSCATSPACE
USERSP1
USERSP2
USERSP3
Full table space backup (BK1)
Full table space backup (BK2)
Full table space backup (BK3)
BK1
BK2
BK3

Figure 23. Backup images available for database SAMPLE

The following procedure demonstrates using the RESTORE DATABASE and ROLLFORWARD DATABASE commands, issued through the CLP, to rebuild just SYSCATSPACE and USERSP1 to end of logs:

```
db2 restore db mydb rebuild with all tablespaces in image
        taken at BK1 without prompting
db2 rollforward db mydb to end of logs
db2 rollforward db mydb stop
```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in restore-pending state. You can still restore USERSP2 and USERSP3 at a later time.

Rebuild and incremental backup images

You can rebuild a database using incremental images. By default, the restore utility tries to use automatic incremental restore for all incremental images. This means that if you do not use the INCREMENTAL option of the RESTORE DATABASE command, but the target image is an incremental backup image, the restore utility will issue the rebuild operation using automatic incremental restore. If the target image is not an incremental image, but another required image is an incremental image then the restore utility will make sure those incremental images are restored using automatic incremental restore. The restore utility will behave in the same way whether you specify the INCREMENTAL option with the AUTOMATIC option or not.

If you specify the INCREMENTAL option but not the AUTOMATIC option, you will need to perform the entire rebuild process manually. The restore utility will just restore the initial metadata from the target image, as it would in a regular manual incremental restore. You will then need to complete the restore of the target image using the required incremental restore chain. Then you will need to restore the remaining images to rebuild the database.

It is recommended that you use automatic incremental restore to rebuild your database. Only in the event of a restore failure, should you attempt to rebuild a database using manual methods.

Rebuilding partitioned databases

To rebuild a partitioned database, rebuild each database partition separately. For each database partition, beginning with the catalog partition, first restore all the table spaces that you require. Any table spaces that are not restored are placed in restore pending state. Once all the database partitions are restored, you then issue the ROLLFORWARD DATABASE command on the catalog partition to roll all of the database partitions forward.

Note: If, at a later date, you need to restore any table spaces that were not originally included in the rebuild phase, you need to make sure that when you subsequently roll the table space forward that the rollforward utility keeps all the data across the database partitions synchronized. If a table space is missed during the original restore and rollforward operation, it might not be detected until there is an attempt to access the data and a data access error occurs. You will then need to restore and roll the missing table space forward to get it back in sync with the rest of the partitions.

To rebuild a partitioned database using table-space level backup images, consider the following example.

In this example, there is a recoverable database called SAMPLE with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BKxy represents backup number x on partition y:

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2
- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

The following procedure demonstrates using the RESTORE DATABASE and ROLLFORWARD DATABASE commands, issued through the CLP, to rebuild the entire database to the end of logs.

1. On database partition 1, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db sample rebuild with all tablespaces in database
      taken at BK31 without prompting
   ```
2. On database partition 2, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db sample rebuild with tablespaces in database
      taken at BK42 without prompting
   ```
3. On database partition 3, issue a RESTORE DATABASE command with the REBUILD option:

   ```
   db2 restore db sample rebuild with all tablespaces in database
      taken at BK43 without prompting
   ```
4. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option:

   ```
   db2 rollforward db sample to end of logs
   ```
5. Issue a ROLLFORWARD DATABASE command with the STOP option:

   ```
   db2 rollforward db sample stop
   ```

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Restrictions for database rebuild

The following list is a summary of database rebuild restrictions:

- One of the table spaces you rebuild must be SYSCATSPACE on the catalog partition.
- You cannot perform a rebuild operation using the Control Center GUI tools. You must either issue commands using the command line processor (CLP) or use the corresponding application programming interfaces (APIs).
- The REBUILD option cannot be used against a pre-Version 9.1 target image unless the image is that of an offline database backup. If the target image is an offline database backup then only the table spaces in this image can be used for the rebuild. The database will need to be migrated after the rebuild operation successfully completes. Attempts to rebuild using any other type of pre-Version 9.1 target image will result in an error.
- The REBUILD option cannot be issued against a target image from a different operating system than the one being restored on unless the target image is a full database backup. If the target image is a full database backup then only the table spaces in this image can be used for the rebuild. Attempts to rebuild using any other type of target image from a different operating system than the one being restored on will result in an error.

Optimizing restore performance

When you perform a restore operation, DB2 will automatically choose an optimal value for the number of buffers, the buffer size and the parallelism settings. The values will be based on the amount of utility heap memory available, the number of processors available and the database configuration. Therefore, depending on the amount of storage available on your system, you should consider allocating more memory by increasing the UTIL_HEAP_SZ configuration parameter. The objective is to minimize the time it takes to complete a restore operation. Unless you explicitly enter a value for the following RESTORE DATABASE command parameters, DB2 will select one for them:

- WITH num-buffers BUFFERS

- PARALLELISM n
- BUFFER buffer-size

For restore operations, a multiple of the buffer size used by the backup operation will always be used. You can specify a buffer size when you issue the RESTORE DATABASE command but you need to make sure that it is a multiple of the backup buffer size.

You can also choose to do any of the following to reduce the amount of time required to complete a restore operation:

- Increase the restore buffer size.

  The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers allocated will be the smallest acceptable size.
- Increase the number of buffers.

  The value you specify must be a multiple of the number of pages that you specified for the backup buffer. The minimum number of pages is 8.
- Increase the value of the PARALLELISM parameter.

  This will increase the number of buffer manipulators (BM) that will be used to write to the database during the restore operation.
- Increase the utility heap size

  This will increase the memory that can be used simultaneously by the other utilities.

Privileges, authorities, and authorization required to use restore

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to restore to an *existing* database from a full database backup. To restore to a *new* database, you must have SYSADM or SYSCTRL authority.

Restore examples

Redirected Restore sessions - CLP examples

Example 1

Following is a typical non-incremental redirected restore scenario for a database whose alias is MYDB:

1. Issue a RESTORE DATABASE command with the REDIRECT option.
   ```
   db2 restore db mydb replace existing redirect
   ```
2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers you want to redefine. For example, in a Windows environment:
   ```
   db2 set tablespace containers for 5 using
         (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command for every table space whose container locations are being redefined.

3. After successful completion of steps 1 and 2, issue:
   ```
   db2 restore db mydb continue
   ```
   This is the final step of the redirected restore operation.
4. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:
   ```
   db2 restore db mydb abort
   ```
2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

Example 2

Following is a typical manual incremental redirected restore scenario for a database whose alias is MYDB and has the following backup images:

```
backup db mydb
Backup successful. The timestamp for this backup image is : <ts1>

backup db mydb incremental
Backup successful. The timestamp for this backup image is : <ts2>
```

1. Issue a RESTORE DATABASE command with the INCREMENTAL and REDIRECT options.
   ```
   db2 restore db mydb incremental taken at <ts2> replace existing redirect
   ```
2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:
   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```
   To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.
3. After successful completion of steps 1 and 2, issue:
   ```
   db2 restore db mydb continue
   ```
4. The remaining incremental restore commands can now be issued as follows:
   ```
   db2 restore db mydb incremental taken at <ts1>
   db2 restore db mydb incremental taken at <ts2>
   ```
   This is the final step of the redirected restore operation.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:
   ```
   db2 restore db mydb abort
   ```
2. After successful completion of step 3, and before issuing all the required commands in step 4, the restore operation can be aborted by issuing:
   ```
   db2 restore db mydb incremental abort
   ```
3. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

4. If either restore command fails in step 4, the failing command can be reissued to continue the restore process.

Example 3

Following is a typical automatic incremental redirected restore scenario for the same database:

1. Issue a RESTORE DATABASE command with the INCREMENTAL AUTOMATIC and REDIRECT options.
   ```
   db2 restore db mydb incremental automatic taken at <ts2>
          replace existing redirect
   ```
2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example, in a Windows environment:
   ```
   db2 set tablespace containers for 5 using
          (file 'f:\ts3con1'20000, file 'f:\ts3con2'20000)
   ```

   To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.
3. After successful completion of steps 1 and 2, issue:
   ```
   db2 restore db mydb continue
   ```

   This is the final step of the redirected restore operation.

Note:

1. After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:
   ```
   db2 restore db mydb abort
   ```
2. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1 after issuing:
   ```
   db2 restore db mydb incremental abort
   ```

Rebuild sessions - CLP examples

Scenario 1

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

Example 1

The following rebuilds the entire database to the most recent point in time:

1. Issue a RESTORE DATABASE command with the REBUILD option:
   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 2

The following rebuilds just SYSCATSPACE and USERSP2 to a point in time (where end of BK3 is less recent than the point in time, which is less recent than end of logs):

1. Issue a RESTORE DATABASE command with the REBUILD option and specify the table spaces you want to include.
   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      taken at BK2 without prompting
   ```
2. Issue a ROLLFORWARD DATABASE command with the TO PIT option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to PIT
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

To restore USERSP1 and USERSP3 at a later time, using normal table space restores (without the REBUILD option):

1. Issue the RESTORE DATABASE command *without* the REBUILD option and specify the table space you want to restore. First restore USERSPI:
   ```
   db2 restore db mydb tablespace (USERSP1) taken at BK1 without prompting
   ```
2. Then restore USERSP3:
   ```
   db2 restore db mydb tablespace taken at BK3 without prompting
   ```
3. Issue a ROLLFORWARD DATABASE command with the END OF LOGS option and specify the table spaces to be restored (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs tablespace (USERSP1, USERSP3)
   ```
   The rollforward will replay all logs up to the PIT and then stop for these two table spaces since no work has been done on them since the first rollforward.
4. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

Example 3

The following rebuilds just SYSCATSPACE and USERSP1 to end of logs:

1. Issue a RESTORE DATABASE command with the REBUILD option:
   ```
   db2 restore db mydb rebuild with all tablespaces in image
      taken at BK1 without prompting
   ```
2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```

3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP1 are in NORMAL state. USERSP2 and USERSP3 are in RESTORE_PENDING state.

Example 4

In the following example, the backups BK1 and BK2 are no longer in the same location as stated in the history file, but this is not known when the rebuild is issued.

1. Issue a RESTORE DATABASE command with the REBUILD option , specifying that you want to rebuild the entire database to the most recent point in time:
   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 without prompting
   ```

   At this point, the target image is restored successfully, but an error is returned from the restore utility stating it could not find a required image.
2. You must now complete the rebuild manually. Since the database is in the rebuild phase this can be done as follows:
   a. Issue a RESTORE DATABASE command and specify the location of the BK1 backup image:
      ```
      db2 restore db mydb tablespace taken at BK1 from <location>
         without prompting
      ```
   b. Issue a RESTORE DATABASE command and specify the location of the BK2 backup image:
      ```
      db2 restore db mydb tablespace (USERSP2) taken at BK2 from
         <location> without prompting
      ```
   c. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
      ```
      db2 rollforward db mydb to end of logs
      ```
   d. Issue a ROLLFORWARD DATABASE command with the STOP option:
      ```
      db2 rollforward db mydb stop
      ```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 5

In this example, table space USERSP3 contains independent data that is needed for generating a specific report, but you do not want the report generation to interfere with the original database. In order to gain access to the data but not affect the original database, you can use REBUILD to generate a new database with just this table space and SYSCATSPACE. SYSCATSPACE is also required so that the database will be connectable after the restore and roll forward operations.

To build a new database with the most recent data in SYSCATSPACE and USERSP3:

1. Issue a RESTORE DATABASE command with the REBUILD option, and specify that table spaces SYSCATSPACE and USERSP3 are to be restored to a new database, NEWDB:
   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP3)
      taken at BK3 into newdb without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command on NEWDB with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db newdb to end of logs
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db newdb stop
   ```

At this point the new database is connectable and only SYSCATSPACE and USERSP3 are in NORMAL state. USERSP1 and USERSP2 are in RESTORE_PENDING state.

Note: If container paths are an issue between the current database and the new database (for example, if the containers for the original database need to be altered because the file system does not exist or if the containers are already in use by the original database) then you will need to perform a redirected restore. The example above assumes the default autostorage database paths are used for the table spaces.

Scenario 2

In the following example, there is a recoverable database called MYDB that has SYSCATSPACE and one thousand user table spaces named Txxxx, where x stands for the table space number (for example, T0001). There is one full database backup image (BK1)

Example 6

The following restores all table spaces except T0999 and T1000:
1. Issue a RESTORE DATABASE command with the REBUILD option:
   ```
   db2 restore db mydb rebuild with all tablespaces in image except
      tablespace (T0999, T1000) taken at BK1 without prompting
   ```
2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database will be connectable and all table spaces except T0999 and T1000 will be in NORMAL state. T0999 and T1000 will be in RESTORE_PENDING state.

Scenario 3

The examples in this scenario demonstrate how to rebuild a recoverable database using incremental backups. In the following examples, there is a database called MYDB with the following table spaces in it:
- SYSCATSPACE (system catalogs)
- USERSP1 (data table space)
- USERSP2 (user data table space)
- USERSP3 (user data table space)

The following backups have been taken:
- FULL1 is a full backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA1 is a delta backup of SYSCATSPACE and USERSP1

- INCR1 is an incremental backup of USERSP2 and USERSP3
- DELTA2 is a delta backup of SYSCATSPACE, USERSP1, USERSP2 and USERSP3
- DELTA3 is a delta backup of USERSP2
- FULL2 is a full backup of USERSP1

Example 7

The following rebuilds just SYSCATSPACE and USERSP2 to the most recent point in time using incremental automatic restore.

1. Issue a RESTORE DATABASE command with the REBUILD option. The INCREMENTAL AUTO option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.
   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      incremental auto taken at DELTA3 without prompting
   ```
2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 and USERSP3 are in RESTORE_PENDING state.

Example 8

The following rebuilds the entire database to the most recent point in time using incremental automatic restore.

1. Issue a RESTORE DATABASE command with the REBUILD option. The INCREMENTAL AUTO option is optional. The restore utility will detect what the granularity of the image is and use automatic incremental restore if it is required.
   ```
   db2 restore db mydb rebuild with all tablespaces in database
      incremental auto taken at DELTA3 without prompting
   ```
2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Example 9

The following rebuilds the entire database, except for USERSP3, to the most recent point in time.

1. Issue a RESTORE DATABASE command with the REBUILD option. Although the target image is a non-incremental image, the restore utility will detect that the required rebuild chain includes incremental images and it will automatically restore those images incrementally.
   ```
   db2 restore db mydb rebuild with all tablespaces in database except
      tablespace (USERSP3) taken at FULL2 without prompting
   ```

2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

Scenario 4

The examples in this scenario demonstrate how to rebuild a recoverable database using backup images that contain log files. In the following examples, there is a database called MYDB with the following table spaces in it:
- SYSCATSPACE (system catalogs)
- USERSP1 (user data table space)
- USERSP2 (user data table space)

Example 10

The following rebuilds the database with just SYSCATSPACE and USERSP2 to the most recent point in time. There is a full online database backup image (BK1), which includes log files.
1. Issue a RESTORE DATABASE command with the REBUILD option:
   ```
   db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP2)
      taken at BK1 logtarget /logs without prompting
   ```
2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs after the end of BK1 have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs overflow log path (/logs)
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and only SYSCATSPACE and USERSP2 are in NORMAL state. USERSP1 is in RESTORE_PENDING state.

Example 11

The following rebuilds the database to the most recent point in time. There are two full online table space backup images that include log files:
- BK1 is a backup of SYSCATSPACE, using log files 10-45
- BK2 is a backup of USERSP1 and USERSP2, using log files 64-80

1. Issue a RESTORE DATABASE command with the REBUILD option:
   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK2 logtarget /logs without prompting
   ```

   The rollforward operation will start at log file 10, which it will always find in the overflow log path if not in the primary log file path. The log range 46-63, since they are not contained in any backup image, will need to be made available for roll forward.
2. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option, using the overflow log path for log files 64-80:
   ```
   db2 rollforward db mydb to end of logs overflow log path (/logs)
   ```
3. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 5

In the following examples, there is a recoverable database called MYDB with the following table spaces in it:

- SYSCATSPACE (0), SMS system catalog (relative container)
- USERSP1 (1) SMS user data table space (relative container)
- USERSP2 (2) DMS user data table space (absolute container /usersp2)
- USERSP3 (3) DMS user data table space (absolute container /usersp3)

The following backups have been taken:

- BK1 is a backup of SYSCATSPACE and USERSP1
- BK2 is a backup of USERSP2 and USERSP3
- BK3 is a backup of USERSP3

Example 12

The following rebuilds the entire database to the most recent point in time using redirected restore.

1. Issue a RESTORE DATABASE command with the REBUILD option:
   ```
   db2 restore db mydb rebuild with all tablespaces in database
      taken at BK3 redirect without prompting
   ```
2. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers you want to redefine. For example:
   ```
   db2 set tablespace containers for 3 using (file '/newusersp2' 10000)
   ```
3.
   ```
   db2 set tablespace containers for 4 using (file '/newusersp3' 15000)
   ```
4. Issue a RESTORE DATABASE command with the CONTINUE option:
   ```
   db2 restore db mydb continue
   ```
5. Issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (this assumes all logs have been saved and are accessible):
   ```
   db2 rollforward db mydb to end of logs
   ```
6. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

At this point the database is connectable and all table spaces are in NORMAL state.

Scenario 6

In the following examples, there is a database called MYDB with three database partitions:

- Database partition 1 contains table spaces SYSCATSPACE, USERSP1 and USERSP2, and is the catalog partition
- Database partition 2 contains table spaces USERSP1 and USERSP3
- Database partition 3 contains table spaces USERSP1, USERSP2 and USERSP3

The following backups have been taken, where BKxy represents backup number x on partition y:

- BK11 is a backup of SYSCATSPACE, USERSP1 and USERSP2
- BK12 is a backup of USERSP2 and USERSP3
- BK13 is a backup of USERSP1, USERSP2 and USERSP3
- BK21 is a backup of USERSP1
- BK22 is a backup of USERSP1
- BK23 is a backup of USERSP1
- BK31 is a backup of USERSP2
- BK33 is a backup of USERSP2
- BK42 is a backup of USERSP3
- BK43 is a backup of USERSP3

Example 13

The following rebuilds the entire database to the end of logs.

1. On database partition 1, issue a RESTORE DATABASE command with the REBUILD option:
```
db2 restore db mydb rebuild with all tablespaces in database
   taken at BK31 without prompting
```
2. On database partition 2, issue a RESTORE DATABASE command with the REBUILD option:
```
db2 restore db mydb rebuild with tablespaces in database taken at
   BK42 without prompting
```
3. On database partition 3, issue a RESTORE DATABASE command with the REBUILD option:
```
db2 restore db mydb rebuild with all tablespaces in database
   taken at BK43 without prompting
```
4. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option (assumes all logs have been saved and are accessible on all database partitions):
```
db2 rollforward db mydb to end of logs
```
5. Issue a ROLLFORWARD DATABASE command with the STOP option:
```
db2 rollforward db mydb stop
```

At this point the database is connectable on all database partitions and all table spaces are in NORMAL state.

Example 14

The following rebuilds SYSCATSPACE, USERSP1 and USERSP2 to the most recent point in time.

1. On database partition 1, issue a RESTORE DATABASE command with the REBUILD option:
```
db2 restore db mydb rebuild with all tablespaces in database
   taken at BK31 without prompting
```
2. On database partition 2, issue a RESTORE DATABASE command with the REBUILD option:
```
db2 restore db mydb rebuild with all tablespaces in image taken at
   BK22 without prompting
```
3. On database partition 3, issue a RESTORE DATABASE command with the REBUILD option:

```
db2 restore db mydb rebuild with all tablespaces in image taken at
   BK33 without prompting
```

Note: this command omitted USERSP1, which is needed to complete the rebuild operation.

4. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option:
   ```
   db2 rollforward db mydb to end of logs
   ```
5. Issue a ROLLFORWARD DATABASE command with the STOP option:
   ```
   db2 rollforward db mydb stop
   ```

   The rollforward succeeds and the database is connectable on all database partitions. All table spaces are in NORMAL state, except USERSP3, which is in RESTORE PENDING state on all database partitions on which it exists, and USERSP1, which is in RESTORE PENDING state on database partition 3.

   When an attempt is made to access data in USERSP1 on database partition 3, a data access error will occur. To fix this, USERSP1 will need to be recovered:
   a. On database partitions 3, issue a RESTORE DATABASE command, specifying a backup image that contains USERSP1:
      ```
      db2 restore db mydb tablespace taken at BK23 without prompting
      ```
   b. On the catalog partition, issue a ROLLFORWARD DATABASE command with the TO END OF LOGS option and the AND STOP option:
      ```
      db2 rollforward db mydb to end of logs on dbpartitionnum (3) and stop
      ```

At this point USERSP1 on database partition 3 can have its data accessed since it is in NORMAL state.

Scenario 7

In the following examples, there is a *nonrecoverable* database called MYDB with the following table spaces:
- SYSCATSPACE (0), SMS system catalog
- USERSP1 (1) SMS user data table space
- USERSP2 (2) DMS user data table space
- USERSP3 (3) DMS user data table space

There is just one backup of the database, BK1:

Example 15

The following demonstrates using rebuild on a nonrecoverable database.

Rebuild the database using only SYSCATSPACE and USERSP1:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1)
   taken at BK1 without prompting
```

Following the restore, the database is connectable. If you issue the LIST TABLESPACES command you see that that SYSCATSPACE and USERSP1 are in NORMAL state, while USERSP2 and USERSP3 are in DELETE_PENDING/ OFFLINE state. You can now work with the two table spaces that are in NORMAL state.

If you want to do a database backup, you will first need to drop USERSP2 and USERSP3 using the DROP TABLESPACE command, otherwise, the backup will fail.

To restore USERSP2 and USERSP3 at a later time, you need to reissue a database restore from BK1.

Chapter 14. Rollforward overview

The simplest form of the DB2 ROLLFORWARD DATABASE command requires only that you specify the alias name of the database that you want to rollforward recover. For example:

```
   db2 rollforward db sample to end of logs and stop
```

In this example, the command returns:

```
                                 Rollforward Status

 Input database alias                   = sample
 Number of nodes have returned status   = 1

 Node number                            = 0
 Rollforward status                     = not pending
 Next log file to be read               =
 Log files processed                    =  -
 Last committed transaction             = 2001-03-11-02.39.48.000000

DB20000I  The ROLLFORWARD command completed successfully.
```

The following is one approach you can use to perform rollforward recovery:

1. Invoke the rollforward utility without the STOP option.
2. Invoke the rollforward utility with the QUERY STATUS option

   If you specify recovery to the end of the logs, the QUERY STATUS option can indicate that one or more log files is missing, if the returned point in time is earlier than you expect.

   If you specify point-in-time recovery, the QUERY STATUS option will help you to ensure that the rollforward operation has completed at the correct point.
3. Invoke the rollforward utility with the STOP option. After the operation stops, it is not possible to roll additional changes forward.

An alternate approach you can use to perform rollforward recovery is the following:

1. Invoke the rollforward utility with the AND STOP option.
2. The need to take further steps depends on the outcome of the rollforward operation:
   - If it is successful, the rollforward is complete and the database will be connectable and usable. At this point it is not possible to roll additional changes forward.
   - If any errors were returned, take whatever action is required to fix the problem (for example, if there is a missing logfile, find the log file, or if there are retrieve errors, ensure that log archiving is working). Then reissue the rollforward utility with the AND STOP option.

A database must be restored successfully (using the restore utility) before it can be rolled forward, but a table space does not. A table space can be temporarily put in rollforward pending state, but not require a restore operation to undo it (following a power interruption, for example).

When the rollforward utility is invoked:

- If the database is in rollforward pending state, the database is rolled forward. If table spaces are also in rollforward pending state, you must invoke the rollforward utility again after the database rollforward operation completes to roll the table spaces forward.
- If the database is *not* in rollforward pending state, but table spaces in the database *are* in rollforward pending state:
  - If you specify a list of table spaces, only those table spaces are rolled forward.
  - If you do not specify a list of table spaces, all table spaces that are in rollforward pending state are rolled forward.

A database rollforward operation runs offline. The database is not available for use until the rollforward operation completes successfully, and the operation cannot complete unless the STOP option was specified when the utility was invoked.

A table space rollforward operation can run offline. The database is not available for use until the rollforward operation completes successfully. This occurs if the end of the logs is reached, or if the STOP option was specified when the utility was invoked.

You can perform an *online* rollforward operation on table spaces, as long as SYSCATSPACE is not included. When you perform an online rollforward operation on a table space, the table space is not available for use, but the other table spaces in the database *are* available.

When you first create a database, it is enabled for circular logging only. This means that logs are reused, rather than being saved or archived. With circular logging, rollforward recovery is not possible: only crash recovery or version recovery can be done. Archived logs document changes to a database that occur after a backup was taken. You enable log archiving (and rollforward recovery) by setting the *logarchmeth1* database configuration parameter to a value other than its default of `OFF`. When you set *logarchmeth1* to a value other than `OFF`, the database is placed in backup pending state, and you must take an offline backup of the database before it can be used again.

Note: Entries will be made in the recovery history file for each log file that is used in a rollforward operation.

Using rollforward

Use the ROLLFORWARD DATABASE command to apply transactions that were recorded in the database log files to a restored database backup image or table space backup image.

You should not be connected to the database that is to be rollforward recovered: the rollforward utility automatically establishes a connection to the specified database, and this connection is terminated at the completion of the rollforward operation.

Do not restore table spaces without cancelling a rollforward operation that is in progress; otherwise, you might have a table space set in which some table spaces are in rollforward in progress state, and some table spaces are in rollforward pending state. A rollforward operation that is in progress will only operate on the tables spaces that are in rollforward in progress state.

The database can be local or remote.

The following restrictions apply to the rollforward utility:

- You can invoke only one rollforward operation at a time. If there are many table spaces to recover, you can specify all of them in the same operation.
- If you have renamed a table space following the most recent backup operation, ensure that you use the new name when rolling the table space forward. The previous table space name will not be recognized.
- You cannot cancel a rollforward operation that is running. You can only cancel a rollforward operation that has completed, but for which the STOP option has not been specified, or a rollforward operation that has failed before completing.
- You cannot *continue* a table space rollforward operation to a point in time, specifying a time stamp that is less than the previous one. If a point in time is not specified, the previous one is used. You can issue a rollforward operation that ends at a specified point in time by just specifying STOP, but this is only allowed if the table spaces involved were all restored from the same offline backup image. In this case, no log processing is required. If you start another rollforward operation with a different table space list before the in-progress rollforward operation is either completed or cancelled, an error message (SQL4908) is returned. Invoke the LIST TABLESPACES command on all database partitions to determine which table spaces are currently being rolled forward (rollforward in progress state), and which table spaces are ready to be rolled forward (rollforward pending state). You have three options:
  - Finish the in-progress rollforward operation on all table spaces.
  - Finish the in-progress rollforward operation on a subset of table spaces. (This might not be possible if the rollforward operation is to continue to a specific point in time, which requires the participation of all database partitions.)
  - Cancel the in-progress rollforward operation.
- In a partitioned database environment, the rollforward utility must be invoked from the catalog partition of the database.
- Point in time rollforward of a table space is available only from DB2 Version 9 clients. You should upgrade any clients running an earlier version of the database product to Version 9 in order to roll a table space forward to a point in time.
- You cannot roll forward logs from a previous release version.

The rollforward utility can be invoked through the command line processor (CLP), the Restore wizard in the Control Center, or the db2Rollforward application programming interface (API).

Following is an example of the ROLLFORWARD DATABASE command issued through the CLP:

```
db2 rollforward db sample to end of logs and stop
```

To open the Restore wizard:

1. From the Control Center, expand the object tree until you find the database or table space object that you want to restore.
2. Right-click on the object and select Roll-forward from the pop-up menu. The Rollforward wizard opens.

Detailed information is provided through the contextual help facility within the Control Center.

Rolling forward changes in a table space

If the database is enabled for forward recovery, you have the option of backing up, restoring, and rolling forward table spaces instead of the entire database. You might want to implement a recovery strategy for individual table spaces because this can save time: it takes less time to recover a portion of the database than it does to recover the entire database. For example, if a disk is bad, and it contains only one table space, that table space can be restored and rolled forward without having to recover the entire database, and without impacting user access to the rest of the database, unless the damaged table space contains the system catalog tables; in this situation, you cannot connect to the database. (The system catalog table space can be restored independently if a table space-level backup image containing the system catalog table space is available.) Table space-level backups also allow you to back up critical parts of the database more frequently than other parts, and requires less time than backing up the entire database.

After a table space is restored, it is always in rollforward pending state. To make the table space usable, you must perform rollforward recovery on it. In most cases, you have the option of rolling forward to the end of the logs, or rolling forward to a point in time. You cannot, however, roll table spaces containing system catalog tables forward to a point in time. These table spaces must be rolled forward to the end of the logs to ensure that all table spaces in the database remain consistent.

When a table space is rolled forward, DB2 will process all log files even if they do not contain log records that affect that table space. To skip the log files known not to contain any log records affecting the table space, set the DB2_COLLECT_TS_REC_INFO registry variable to *ON*. This is the default value. To ensure that the information required for skipping log files is collected, the registry variable must be set before the log files are created and used.

The table space change history file (DB2TSCHG.HIS), located in the database directory, keeps track of which logs should be processed for each table space. You can view the contents of this file using the db2logsForRfwd utility, and delete entries from it using the PRUNE HISTORY command. During a database restore operation, DB2TSCHG.HIS is restored from the backup image and then brought up to date during the database rollforward operation. If no information is available for a log file, it is treated as though it is required for the recovery of every table space.

Since information for each log file is flushed to disk after the log becomes inactive, this information can be lost as a result of a crash. To compensate for this, if a recovery operation begins in the middle of a log file, the entire log is treated as though it contains modifications to every table space in the system. After this, the active logs will be processed and the information for them will be rebuilt. If information for older or archived log files is lost in a crash situation and no information for them exists in the data file, they will be treated as though they contain modifications for every table space during the table space recovery operation.

Before rolling a table space forward, invoke the LIST TABLESPACES SHOW DETAIL command. This command returns the *minimum recovery time*, which is the earliest point in time to which the table space can be rolled forward. The minimum recovery time is updated when data definition language (DDL) statements are run against the table space, or against tables in the table space. The table space must be rolled forward to at least the minimum recovery time, so that it becomes synchronized with the information in the system catalog tables. If recovering more than one table space, the table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces being recovered. In a partitioned database environment, issue the LIST TABLESPACES SHOW DETAIL command on all database partitions. The table spaces must be rolled forward to at least the highest minimum recovery time of all the table spaces on all database partitions.

If you are rolling table spaces forward to a point in time, and a table is contained in multiple table spaces, all of these table spaces must be rolled forward simultaneously. If, for example, the table data is contained in one table space, and the index for the table is contained in another table space, you must roll both table spaces forward simultaneously to the same point in time.

If the data and the long objects in a table are in separate table spaces, and the long object data has been reorganized, the table spaces for both the data and the long objects must be restored and rolled forward together. You should take a backup of the affected table spaces after the table is reorganized.

If you want to roll a table space forward to a point in time, and a table in the table space is either:

- An underlying table for a materialized query or staging table that is in another table space
- A materialized query or staging table for a table in another table space

You should roll both table spaces forward to the same point in time. If you do not, the materialized query or staging table is placed in set integrity pending state at the end of the rollforward operation. The materialized query table will need to be fully refreshed, and the staging table will be marked as incomplete.

If you want to roll a table space forward to a point in time, and a table in the table space participates in a referential integrity relationship with another table that is contained in another table space, you should roll both table spaces forward simultaneously to the same point in time. If you do not, the child table in the referential integrity relationship will be placed in set integrity pending state at the end of the rollforward operation. When the child table is later checked for constraint violations, a check on the entire table is required. If any of the following tables exist, they will also be placed in set integrity pending state with the child table:

- Any descendent materialized query tables for the child table
- Any descendent staging tables for the child table
- Any descendent foreign key tables of the child table

These tables will require full integrity processing to bring them out of the set integrity pending state. If you roll both table spaces forward simultaneously, the constraint will remain active at the end of the point-in-time rollforward operation.

Ensure that a point-in-time table space rollforward operation does not cause a transaction to be rolled back in some table spaces, and committed in others. This can happen if:

- A point-in-time rollforward operation is performed on a subset of the table spaces that were updated by a transaction, and that point in time precedes the time at which the transaction was committed.
- Any table contained in the table space being rolled forward to a point in time has an associated trigger, or is updated by a trigger that affects table spaces other than the one that is being rolled forward.

The solution is to find a suitable point in time that will prevent this from happening.

You can issue the QUIESCE TABLESPACES FOR TABLE command to create a transaction-consistent point in time for rolling table spaces forward. The quiesce request (in share, intent to update, or exclusive mode) waits (through locking) for all running transactions against those table spaces to complete, and blocks new requests. When the quiesce request is granted, the table spaces are in a consistent state. To determine a suitable time to stop the rollforward operation, you can look in the recovery history file to find quiesce points, and check whether they occur after the minimum recovery time.

After a table space point-in-time rollforward operation completes, the table space is put in backup pending state. You must take a backup of the table space, because all updates made to it between the point in time to which you rolled forward and the current time have been removed. You can no longer roll the table space forward to the current time from a previous database- or table space-level backup image. The following example shows why the table space-level backup image is required, and how it is used. (To make the table space available, you can either back up the entire database, the table space that is in backup pending state, or a set of table spaces that includes the table space that is in backup pending state.)

```
Database                             Time of rollforward of    Restore
backup                               table space TABSP1 to     database.
                                     T2. Back up TABSP1.       Roll forward
                                                               to end of logs.
T1             T2                    T3                        T4
|              |                     |                         |
|              |                     |                         |
|----------------------------------------------------------------------------
               | Logs are not
                 applied to TABSP1
                 between T2 and T3
                 when it is rolled
                 forward to T2.
```

Figure 24. Table Space Backup Requirement

In the preceding example, the database is backed up at time T1. Then, at time T3, table space TABSP1 is rolled forward to a specific point in time (T2), The table space is backed up after time T3. Because the table space is in backup pending state, this backup operation is mandatory. The time stamp of the table space backup image is after time T3, but the table space is at time T2. Log records from between T2 and T3 are not applied to TABSP1. At time T4, the database is restored, using the backup image created at T1, and rolled forward to the end of the logs. Table space TABSP1 is put in restore pending state at time T3, because the database manager assumes that operations were performed on TABSP1 between T3 and T4 without the log changes between T2 and T3 having been applied to the table space. If these log changes were in fact applied as part of the rollforward operation against the database, this assumption would be incorrect. The table space-level backup that must be taken after the table space is rolled forward to a point in time allows you to roll that table space forward past a previous point-in-time rollforward operation (T3 in the example).

Assuming that you want to recover table space TABSP1 to T4, you would restore the table space from a backup image that was taken after T3 (either the required backup, or a later one), then roll TABSP1 forward to the end of the logs.

In the preceding example, the most efficient way of restoring the database to time T4 would be to perform the required steps in the following order:

1. Restore the database.

2. Restore the table space.
3. Roll the database forward.
4. Roll the table space forward.

Because you restore the table space before rolling the database forward, resource is not used to apply log records to the table space when the database is rolled forward.

If you cannot find the TABSP1 backup image that follows time T3, or you want to restore TABSP1 to T3 (or earlier), you can:

- Roll the table space forward to T3. You do not need to restore the table space again, because it was restored from the database backup image.
- Restore the table space again, using the database backup taken at time T1, then roll the table space forward to a time that precedes time T3.
- Drop the table space.

In a partitioned database environment:

- You must simultaneously roll all parts of a table space forward to the same point in time at the same time. This ensures that the table space is consistent across database partitions.
- If some database partitions are in rollforward pending state, and on other database partitions, some table spaces are in rollforward pending state (but the database partitions are not), you must first roll the database partitions forward, and then roll the table spaces forward.
- If you intend to roll a table space forward to the end of the logs, you do not have to restore it at each database partition; you only need to restore it at the database partitions that require recovery. If you intend to roll a table space forward to a point in time, however, you must restore it at each database partition.

In a database with partitioned tables:

- If you are rolling a table space containing any piece of a partitioned table forward to a point in time, you must also roll all of the other table spaces in which that table resides forward to the same point in time. However, rolling a single table space containing a piece of a partitioned table forward to the end of logs is allowed. If a partitioned table has any attached, detached, or dropped data partitions, then point-in-time rollforward must also include all table spaces for these data partitions. In order to determine if a partitioned table has any attached, detached, or dropped data partitions, query the SYSCAT.DATAPARTITIONS catalog view.

Authorization required for rollforward

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the rollforward utility.

Rollforward sessions - CLP examples

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands are:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

```
db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected before you stop it, so that you do not miss any logs.

If the rollforward command encounters an error, the rollforward operation will not complete. The error will be returned, and you will then be able to fix the error and reissue the command. If, however, you are unable to fix the error, you can force the rollforward to complete by issuing the following:

```
db2 rollforward db sample complete
```

This command brings the database online at the point in the logs before the failure.

Example 2

Roll the database forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS2, TBS3) online
```

Note that two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   overflow log path (/logs)
```

Example 5

In the following example, there is a database called sample. The database is backed up and the recovery logs are included in the backup image; the database is restored; and the database is rolled forward to the end of backup timestamp.

Back up the database, including the recovery logs in the backup image:

```
db2 backup db sample online include logs
```

Restore the database using that backup image:

```
db2 restore db sample
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup
```

Example 6 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database partitions 0 and 2. After restoring the database on database partition 1, and TBS1 on database partitions 0 and 2, roll the database forward on database partition 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 ("Database is recovered but one or more table spaces are offline on database partitions 0 and 2.").

```
db2 rollforward db sample to end of logs
```

This rolls TBS1 forward on database partitions 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 7 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with a single system view backup; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Perform a single system view (SSV) backup:

```
db2 backup db sample on all nodes online include logs
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample taken at 1998-04-03-14.21.56.245378"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

Example 8 (partitioned database environments)

In the following example, there is a partitioned database called sample. All the database partitions are backed up with one command using db2_all; the database is restored on all database partitions; and the database is rolled forward to the end of backup timestamp.

Back up all the database partitions with one command using db2_all:

```
db2_all "db2 backup db sample include logs to /shared/dir/"
```

Restore the database on all database partitions:

```
db2_all "db2 restore db sample from /shared/dir/"
```

Roll forward the database to the end of backup timestamp:

```
db2 rollforward db sample to end of backup on all nodes
```

Example 9 (partitioned database environments)

After restoring table space TBS1 on database partitions 0 and 2 only, roll TBS1 forward on database partitions 0 and 2:

```
db2 rollforward db sample to end of logs
```

Database partition 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on
   dbpartitionnums (0, 2) tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together.

Note: With table space rollforward to a point in time, the dbpartitionnum clause is not accepted. The rollforward operation must take place on all the database partitions on which the table space resides.

After restoring TBS1 on database partition 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 10 (partitioned database environments)

After restoring a table space on all database partitions, roll forward to PIT2, but do not specify `AND STOP`. The rollforward operation is still in progress. Cancel and roll forward to PIT1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)
db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all dbpartitionnums **

db2 rollforward db sample to pit1 tablespace(TBS1)
db2 rollforward db sample stop tablespace(TBS1)
```

Example 11 (partitioned database environments)

Rollforward recover a table space that resides on eight database partitions (3 to 10) listed in the `db2nodes.cfg` file:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The database partitions on which the table space resides do not have to be specified. The utility defaults to the db2nodes.cfg file.

Example 12 (partitioned database environments)

Rollforward recover six small table spaces that reside on a single database partition database partition group (on database partition 6):

```
db2 rollforward database dwtest to end of logs on dbpartitionnum (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

Example 13 (Partitioned tables - Rollforward to end of log on all data partitions)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. All table spaces can be rolled forward to END OF LOGS.

```
db2 rollforward db PBARDB to END OF LOGS and stop
   tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 14 (Partitioned tables - Rollforward to end of logs on one table space)

A partitioned table is created initially using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, and attaches data partitions from the table in tbsp5. Table space tbsp4 becomes corrupt and requires a restore and rollforward to end of logs.

```
db2 rollforward db PBARDB to END OF LOGS and stop tablespace(tbsp4)
```

This completes successfully.

Example 15 (Partitioned tables - Rollforward to PIT of all data partitions including those added, attached, detached or with indexes)

A partitioned table is created using table spaces tbsp1, tbsp2, tbsp3 with an index in tbsp0. Later on, a user adds data partitions to the table in tbsp4, attaches data partitions from the table in tbsp5, and detaches data partitions from tbsp1. The user performs a rollforward to PIT with all the table spaces used by the partitioned table including those table spaces specified in the INDEX IN clause.

```
db2 rollforward db PBARDB to 2005-08-05-05.58.53.000000 and stop
   tablespace(tbsp0, tbsp1, tbsp2, tbsp3, tbsp4, tbsp5)
```

This completes successfully.

Example 16 (Partitioned tables - Rollforward to PIT on a subset of the table spaces)

A partitioned table is created using three table spaces (tbsp1, tbsp2, tbsp3). Later, the user detaches all data partitions from tbsp3. The rollforward to PIT is only permitted on tbsp1 and tbsp2.

```
db2 rollforward db PBARDB to 2005-08-05-06.02.42.000000 and stop
   tablespace( tbsp1, tbsp2)
```

This completes successfully.

Chapter 15. Data recovery with IBM Tivoli Storage Manager (TSM)

When calling the BACKUP DATABASE or RESTORE DATABASE commands, you can specify that you want to use the IBM Tivoli Storage Manager (TSM) product to manage database or table space backup or restore operation. The minimum required level of TSM client API is Version 4.2.0, except on the following:

- 64-bit Solaris systems, which require TSM client API Version 4.2.1.
- 64-bit Windows operating systems, which require TSM client API Version 5.1.
- All Windows X64 systems, which require TSM client API Version 5.3.2.
- 32-bit Linux for iSeries® and pSeries, which require TSM client API Version 5.1.5 or later.
- 64-bit Linux for System i and pSeries, which require TSM client API Version 5.2.2 or later.
- 64-bit Linux on AMD Opteron systems, which require TSM client API Version 5.2.0 or later.
- Linux for zSeries, which requires TSM client API Version 5.2.2 or later.

Configuring a Tivoli Storage Manager client

Before the database manager can use the TSM option, the following steps might be required to configure the TSM environment:

1. A functioning TSM client and server must be installed and configured. In addition, the TSM client API must be installed on each DB2 server.
2. Set the environment variables used by the TSM client API:

   DSMI_DIR
   Identifies the user-defined directory path where the API trusted agent file (`dsmtca`) is located.

   DSMI_CONFIG
   Identifies the user-defined directory path to the `dsm.opt` file, which contains the TSM user options. Unlike the other two variables, this variable should contain a fully qualified path and file name.

   DSMI_LOG
   Identifies the user-defined directory path where the error log (`dsierror.log`) will be created.

   Note: In a multi-partition database environment these settings must be specified in the `sqllib/userprofile` directory.
3. If any changes are made to these environment variables and the database manager is running, you should:
   - Stop the database manager using the db2stop command.
   - Start the database manager using the db2start command.
4. Depending on the server's configuration, a Tivoli client might require a password to interface with a TSM server. If the TSM environment is configured to use `PASSWORDACCESS=generate`, the Tivoli client needs to have its password established.

The executable file `dsmapipw` is installed in the `sqllib/adsm` directory of the instance owner. This executable allows you to establish and reset the TSM password.

To execute the `dsmapipw` command, you must be logged in as the local administrator or "root" user. When this command is executed, you will be prompted for the following information:

- *Old password*, which is the current password for the TSM node, as recognized by the TSM server. The first time you execute this command, this password will be the one provided by the TSM administrator at the time your node was registered on the TSM server.
- *New password*, which is the new password for the TSM node, stored at the TSM server. (You will be prompted twice for the new password, to check for input errors.)

Note: Users who invoke the BACKUP DATABASE or RESTORE DATABASE commands do not need to know this password. You only need to run the `dsmapipw` command to establish a password for the initial connection, and after the password has been reset on the TSM server.

Considerations for using Tivoli Storage Manager

To use specific features within TSM, you might be required to give the fully qualified path name of the object using the feature. (Remember that on Windows operating systems, the \ will be used instead of /.) The fully qualified path name of:

- A full database recovery object is: `/<database>/NODEnnnn/FULL_BACKUP.timestamp.seq_no`
- An incremental database recovery object is: `/<database>/NODEnnnn/DB_INCR_BACKUP.timestamp.seq_no`
- An incremental delta database recovery object is: `/<database>/NODEnnnn/DB_DELTA_BACKUP.timestamp.seq_no`
- A full table space recovery object is: `/<database>/NODEnnnn/TSP_BACKUP.timestamp.seq_no`
- An incremental table space recovery object is: `/<database>/NODEnnnn/TSP_INCR_BACKUP.timestamp.seq_no`
- An incremental delta table space recovery object is: `/<database>/NODEnnnn/TSP_DELTA_BACKUP.timestamp.seq_no`

where `<database>` is the database alias name, and `NODEnnnn` is the node number. The names shown in uppercase characters must be entered as shown.

- In the case where you have multiple backup images using the same database alias name, the time stamp and sequence number become the distinguishing part of a fully qualified name. You will need to query TSM to determine which backup version to use.
- If you perform an online backup operation and specify the USE TSM option and the INCLUDE LOGS option, a deadlock can occur if the two processes try to write to the same tape drive at the same time. If you are using a tape drive as a storage device for logs and backup images, you need to define two separate tape pools for TSM, one for the backup image and one for the archived logs.

Chapter 16. DB2 Advanced Copy Services (ACS)

DB2 Advanced Copy Services (ACS) enables you to use the fast copying technology of a storage device to perform the data copying part of backup and restore operations.

In a traditional backup or restore operation, the database manager copies data to or from disk or a storage device using operating system calls. Being able to use the storage device to perform the data copying makes the backup and restore operations much faster. A backup operation that uses DB2 ACS is called a snapshot backup.

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS series

For detailed instructions on the setup and usage of DB2 Advanced Copy Services (ACS), refer to the Tivoli documentation for Advanced Copy Services at: http://publib.boulder.ibm.com/tividd/td/IBMTivoliStorageManagerforAdvancedCopyServices5.3.3.html

Enabling DB2 Advanced Copy Services (ACS)

To use DB2 Advanced Copy Services (ACS), or perform snapshot backup operations, you must install, activate, and configure DB2 ACS.

Before you begin

DB2 ACS is part of the IBM DB2 High Availability (HA) Feature. To use DB2 ACS, you must have a license for the DB2 HA Feature.

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS series

Procedure

1. Install DB2 ACS. See: “Installing DB2 Advanced Copy Services (ACS)” on page 292.

2. Create the database manager instance or instances with which you will use DB2 ACS.

   When you create a new database manager instance, a directory called acs is created in the new instance sqllib directory. Because each database manager instance has an acs directory, you can configure each database manager instance differently.
3. For each database manager instance with which you will use DB2 ACS, perform the following steps:
   a. Activate DB2 ACS. See: "Activating DB2 Advanced Copy Services (ACS)" on page 293.
   b. Configure DB2 ACS. See: "Configuring DB2 Advanced Copy Services (ACS)" on page 293.

Results

After you have enabled DB2 ACS, you can perform snapshot backup operations.

For detailed instructions on the setup and usage of DB2 Advanced Copy Services (ACS), refer to the Tivoli documentation for Advanced Copy Services at: http://publib.boulder.ibm.com/tividd/td/IBMTivoliStorageManagerforAdvancedCopyServices5.3.3.html

Installing DB2 Advanced Copy Services (ACS)

The files and libraries required for DB2 Advanced Copy Services (ACS) are installed by the IBM Data Server installer.

Restrictions

DB2 ACS supports a subset of hardware and operating systems that IBM Data Server supports. For a list of hardware and operating systems that DB2 ACS supports, see: "DB2 Advanced Copy Services (ACS) supported operating systems and hardware" on page 337.

Before you begin

Before installing ACS, you must have the following libraries installed:

On AIX:
- ln -s /opt/freeware/lib/powerpc-ibm-aix5.3.0//libgcc_s.a /usr/lib/libgcc_s.a

On Red Hat Enterprise Linux:
- ln -s libssl.so.0.9.7xxx libssl.so.0.9.7
- ln -s libcrypto.so.0.9.7xxx libcrypto.so.0.9.7
- ln -s libssl.so.0.9.7xxx libssl.so
- ln -s libssl.so.0.9.7xxx libssl.so.0

Procedure
1. Install IBM Data Server.
2. Add a port for the DB2 ACS agent in the TCP/IP services file. For example:
```
db2acs  5400/tcp # DB2 ACS service port
```

What to do next

After you have installed DB2 ACS, you must activate DB2 ACS and configure DB2 ACS before you can perform snapshot backup operations.

For detailed instructions on the setup and usage of DB2 Advanced Copy Services (ACS), refer to the Tivoli documentation for Advanced Copy Services at: http://publib.boulder.ibm.com/tividd/td/IBMTivoliStorageManagerforAdvancedCopyServices5.3.3.html

Activating DB2 Advanced Copy Services (ACS)

Before you can use DB2 Advanced Copy Services (ACS) to perform a snapshot backup for a given database manager instance, you must activate DB2 ACS functionality on that instance. You activate DB2 ACS by running a script.

Before you begin

You must perform the following tasks before you can activate DB2 ACS:

1. Install DB2 ACS
2. Create the database manager instance or instances with which you will use DB2 ACS.

About this task

The database manager automatically calls setup.sh to activate DB2 ACS functionality during database manager instance creation and when you upgrade IBM Data Server.

You can also activate DB2 ACS manually.

Procedure

To activate DB2 ACS manually, run the setup.sh script as a user with root authority, and with appropriate parameters to activate DB2 ACS.
For more information about setup.sh see: "DB2 Advanced Copy Services (ACS) setup script setup.sh" on page 295.

Results

One important result of running the setup.sh script is that the ownership and permissions of DB2 ACS executable files in the sqllib/acs directory are verified.

What to do next

After you have activated DB2 ACS, you must configure DB2 ACS before you can perform snapshot backup operations.

For detailed instructions on the setup and usage of DB2 Advanced Copy Services (ACS), refer to the Tivoli documentation for Advanced Copy Services at: http://publib.boulder.ibm.com/tividd/td/IBMTivoliStorageManagerforAdvancedCopyServices5.3.3.html

Configuring DB2 Advanced Copy Services (ACS)

Before you can use DB2 Advanced Copy Services (ACS) to perform a snapshot backup, you must configure DB2 ACS. You use configuration files to configure DB2 ACS.

Before you begin

You must perform the following tasks before you can configure DB2 ACS:

1. Install DB2 ACS
2. Create the database manager instance or instances with which you will use DB2 ACS.
3. Activate DB2 ACS

Procedure

Run the setup.sh script from the sqllib/acs directory without any parameters. This will lead you through an interactive, text-based wizard that will configure DB2 ACS. The wizard creates a configuration profile file and modifies the /etc/initab on the machine to trigger the launch of the DB2 ACS daemons.
The following is sample output from the setup.sh wizard:

```
./setup.sh
Do you have a full TSM license to enable all features of TSM for ACS ?[y/n]

n

****** Profile parameters for section GLOBAL: ******
ACS_DIR [/home/krodger/sqllib/acs ]
ACSD [localhost 57328 ]
TRACE [NO ]

****** Profile parameters for section ACSD: ******
ACS_REPOSITORY *mandatory parameter* /home/krodger/acsrepository

****** Profile parameters for section CLIENT: ******
MAX_VERSIONS [ADAPTIVE ] 2
LVM_FREEZE_THAW [YES ]
DEVICE_CLASS [STANDARD ]

****** Profile parameters for section STANDARD: ******
COPYSERVICES_HARDWARE_TYPE *mandatory parameter*
NAS_NSERIES COPYSERVICES_PRIMARY_SERVERNAME *mandatory parameter* fas960a
COPYSERVICES_USERNAME [superuser ] root

======================================================================

The profile has beeen successfully created.
Do you want to continue by specifying passwords for the defined devices? [y/n]

y

Please specify the passwords for the following profile sections:
STANDARD
master

Creating password file at /home/krodger/sqllib/acs/shared/pwd.acsd.
A copy of this file needs to be available to all components that connect to acsd.

BKI1555I: Profile successfully created. Performing additional checks.
Make sure to restart all ACS components to reload the profile.
```

Results

After you have configured DB2 ACS, you can perform snapshot backup operations.

For detailed instructions on the setup and usage of DB2 Advanced Copy Services (ACS), refer to the Tivoli documentation for Advanced Copy Services at:

http://publib.boulder.ibm.com/tividd/td/
IBMTivoliStorageManagerforAdvancedCopyServices5.3.3.html

Configuring the DB2 Advanced Copy Services (ACS) directory

When you create a new database manager instance, a directory called acs is created in the new instance sqllib directory. DB2 Advanced Copy Services (ACS) uses this acs directory to store configuration files like the target volume control file and the shared repository for recovery objects. There are restrictions on the ways you can alter or configure this acs directory.

1. The acs directory must not be involved in any DB2 ACS or snapshot backup operation.
2. The acs directory can be NFS-exported and NFS-shared on all database partitions and on the backup system for a snapshot backup using IBM Tivoli Storage Manager (TSM).

DB2 Advanced Copy Services (ACS) setup script setup.sh

The setup.sh script activates and configures DB2DB2 Advanced Copy Services (ASC).

Location

The script setup.sh is located in the sqllib/acs directory.

Syntax

Here is the syntax for setup.sh:

```
usage: setup.sh -a <action>
                -d <DB2_Instance_Directory>
                -u <Instance_user_ID_name>
                -g <Instance_primary_group_name>
```

where `action` can be one of:

- start
- stop
- query
- enable
- disable

Usage

The database manager automatically calls setup.sh to activate DB2 ACS functionality during database manager instance creation and when you upgrade IBM Data Server.

You can also call the setup.sh script manually:

Activating DB2 ACS

You can activate DB2 ACS by running setup.sh with the parameters described above, as a user with root authority.

Configuring DB2 ACS

You can configure DB2 ACS by running setup.sh without any parameters. If you run setup.sh without any parameters, then a wizard will lead you through DB2 ACS configuration.

One important result of running the setup.sh script is that the ownership and permissions of DB2 ACS executable files in the sqllib/acs directory are verified.

DB2 Advanced Copy Services (ACS) API

The DB2 Advanced Copy Services (ACS) application programming interface (API) defines a set of functions that the database manager uses to communicate with storage hardware to perform snapshot backup operations.

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:
- IBM TotalStorage SAN Volume Controller
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS series

DB2 Advanced Copy Services (ACS) API functions

The database manager communicates DB2 ACS requests to storage hardware through the DB2 ACS API functions.

db2ACSQueryApiVersion - return the current version of the DB2 Advanced Copy Services (ACS) API

Returns the current version of the DB2 Advanced Copy Services (ACS) API.

API include file

```
db2ACSApi.h
```

API and data structure syntax

```
db2ACS_Version db2ACSQueryApiVersion();
```

Parameters

None.

Usage notes

Possible return values:
- `DB2ACS_API_VERSION1`
- `DB2ACS_API_VERSION_UNKNOWN`

db2ACSInitialize - initialize a DB2 Advanced Copy Services (ACS) session

Initializes a new DB2 Advanced Copy Services (ACS) session. This call establishes communication between the database manager's DB2 ACS library and the DB2 ACS API driver for the storage hardware.

Include file

```
db2ACSApi.h
```

Syntax and data structures

```
/* ==========================================================================
 * Session Initialization
 * ========================================================================== */
db2ACS_RC db2ACSInitialize(
               db2ACS_CB               * pControlBlock,
               db2ACS_ReturnCode       * pRC );
```

Parameters

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

Before calling db2ACSInitialize(), the database manager populates the following fields:

pControlBlock->session
pControlBlock->options

The DB2 ACS API driver populates the following fields before returning:

pControlBlock->handle
pControlBlock->vendorInfo

pRC

Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 11. Return codes

Return code	Description	Notes
DB2ACS_RC_OK	The operation was successful.	
DB2ACS_RC_INIT_FAILED	The database manager attempted to initialize a DB2 ACS session, but the initialization failed.	
DB2ACS_RC_INV_ACTION	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_INV_DEV_HANDLE	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

Table 11. Return codes (continued)

Return code	Description	Notes
DB2ACS_RC_DEV_ERROR	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_IO_ERROR	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_COMM_ERROR	There was a communication error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_NO_DEV_AVAIL	There is currently no storage device, such as a tape drive, available to use.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: “DB2 Advanced Copy Services (ACS) API return codes” on page 336.

Usage notes

Before the database manager can make any DB2 ACS API calls, except calls to db2ACSQueryAPIVersion(), the database manager must call db2ACSInitialize(). Once the database manager establishes a DB2 ACS session by calling db2ACSInitialize(), then the database manager can perform any combination of DB2 ACS query, read, write, or delete operations. The database manager can terminate the DB2 ACS session by calling db2ACSTerminate().

db2ACSTerminate - terminate a DB2 Advanced Copy Services (ACS) session

Terminates a DB2 Advanced Copy Services (ACS) session.

Include file

db2ACSApi.h

Syntax and data structures

```
/* ==========================================================================
 * Session Termination
 * ========================================================================== */
db2ACS_RC db2ACSTerminate(
             db2ACS_CB              * pControlBlock,
             db2ACS_ReturnCode      * pRC );
```

Parameters

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

The database manager allocated the memory for this parameter before calling db2ACSInitialize(). The database manager is responsible for freeing this memory after db2ACSTerminate().

Before calling db2ACSTerminate(), the database manager populates the following fields:

pControlBlock->options

The DB2 ACS API driver might invalidate and free the memory in `pControlBlock->vendorInfo.vendorCB`.

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 12. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	Free all memory allocated for this session and terminate.
`DB2ACS_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: “DB2 Advanced Copy Services (ACS) API return codes” on page 336.

Usage notes

The DB2 ACS API driver should free all memory that the driver allocated for the DB2 ACS session in db2ACSTerminate().

Regardless of whether db2ACSTerminate() completes without error, the database manager cannot call any DB2 ACS functions on this DB2 ACS session again, without first calling db2ACSInitialize().

db2ACSPrepare - prepare to perform a snapshot backup operation.

When a snapshot backup is performed, the database manager suspends the database. db2ACSPrepare() performs all the steps to prepare to perform a snapshot backup operation up to, but not including, the point where the database manager suspends the database.

Include file

`db2ACSApi.h`

Syntax and data structures

```
/* ==========================================================================
 * Prepare
 * ========================================================================== */
db2ACS_RC db2ACSPrepare(
             db2ACS_GroupList        * pGroupList,
             db2ACS_CB               * pControlBlock,
             db2ACS_ReturnCode       * pRC );
```

Parameters

pGroupList

Data type: `db2ACS_GroupList *`

`db2ACS_GroupList` contains a list of groups to be included in the snapshot backup operation.

If **pGroupList** is `NULL`, all groups (paths) will be included in the snapshot backup operation.

If **pGroupList** is not `NULL`:

- **pGroupList** contains a list of groups (paths) to be included in the snapshot backup operation.
- The database manager is responsible for allocating and freeing the memory for **pGroupList**.
- The database manager populates the following fields before passing **pGroupList** to db2ACSPrepare():

  pGroupList->numGroupID
  pGroupList->id

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSPrepare(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 13. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: “DB2 Advanced Copy Services (ACS) API return codes” on page 336.

Usage notes

If db2ACSPrepare() succeeds, then the database manager will suspend the database before calling db2ACSSnapshot().

db2ACSBeginOperation - begin a DB2 Advanced Copy Services (ACS) operation.

Begins a DB2 Advanced Copy Services (ACS) operation.

Include file

`db2ACSApi.h`

Syntax and data structures

```
/* ==========================================================================
 * Operation Begin
 *
 * A valid ACS operation is specified by passing an ObjectType OR'd with one of
 * the following Operations, such as:
 *
 *    (DB2ACS_OP_CREATE | DB2ACS_OBJTYPE_SNAPSHOT)
 * ========================================================================== */
db2ACS_RC db2ACSBeginOperation(
               db2ACS_Operation           operation,
               db2ACS_CB                * pControlBlock,
               db2ACS_ReturnCode        * pRC );
```

Parameters

operation

Data type: `db2ACS_Operation`.

operation is a bitmask indicating the DB2 ACS operation to begin, and the type of object involved.

Operation types:

DB2ACS_OP_CREATE
DB2ACS_OP_READ
DB2ACS_OP_DELETE

Object types:

DB2ACS_OBJTYPE_BACKUP
DB2ACS_OBJTYPE_LOG
DB2ACS_OBJTYPE_LOADCOPY
DB2ACS_OBJTYPE_SNAPSHOT

For example: `( DB2ACS_OP_CREATE | DB2ACS_OBJTYPE_SNAPSHOT )` or `( DB2ACS_OP_DELETE | DB2ACS_OBJTYPE_LOADCOPY )`.

The database manager passes **operation** to the db2ACSBeginOperation() function call.

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSBeginOperation(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

If **operation** is `DB2ACS_OP_CREATE` or `DB2ACS_OP_READ`, then the database manager also populates the following field:

pControlBlock->operation

The information contained within `pControlBlock->operation` is only valid within the context of a particular DB2 ACS operation. `pControlBlock->operation` will be set during db2ACSBeginOperation(), and will remain unchanged until db2ACSEndOperation() returns. Neither the database manager nor the DB2 ACS API driver should reference `pControlBlock->operation` outside the scope of a DB2 ACS operation.

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 14. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INV_OPTIONS`	The database manager specified invalid options.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: “DB2 Advanced Copy Services (ACS) API return codes” on page 336.

Usage notes

None.

db2ACSEndOperation - end a DB2 Advanced Copy Services (ACS) operation.

Ends a DB2 Advanced Copy Services (ACS) operation.

Include file

`db2ACSApi.h`

Syntax and data structures

```
/* ==========================================================================
 * Operation End
 * ========================================================================== */
db2ACS_RC db2ACSEndOperation(
              db2ACS_EndAction          endAction,
              db2ACS_CB               * pControlBlock,
              db2ACS_ReturnCode       * pRC );
```

Parameters

endAction

Data type: `db2ACS_EndAction`.

endAction is a bitmask indicating how the DB2 ACS API driver should end the DB2 ACS operation.

Values:

DB2ACS_END_COMMIT
DB2ACS_END_ABORT

The database manager passes **endAction** to the db2ACSEndOperation() function call.

pControlBlock

Data type: `db2ACS_CB`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSEndOperation(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC

Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 15. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	

Table 15. Return codes (continued)

Return code	Description	Notes
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_COMMIT_FAILED`	The DB2 ACS API driver could not commit a transaction.	
`DB2ACS_RC_ABORT_FAILED`	The database manager attempted to abort a DB2 ACS operation, but the attempt to abort failed.	

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

If the database manager passes `DB2ACS_END_ABORT` as the **endAction** parameter, the result should be that the snapshot backup objects are deleted.

db2ACSBeginQuery - begin a query about snapshot backup objects

Begins a DB2 Advanced Copy Services (ACS) query operation about snapshot backup objects that are available to be used for restore operations.

Include file

`db2ACSApi.h`

Syntax and data structures

```
db2ACS_RC db2ACSBeginQuery(
               db2ACS_QueryInput       * pQueryInput,
               db2ACS_CB               * pControlBlock,
               db2ACS_ReturnCode       * pRC );
```

Parameters

pQueryInput

Data type: `db2ACS_QueryInput *`

`db2ACS_QueryInput` has the same fields as `db2ACS_ObjectInfo`. `db2ACS_ObjectInfo` contains information about object created using the DB2 Advanced Copy Services (ACS) API.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

Before calling db2ACSBeginQuery(), the database manager populates the fields of **pQueryInput**.

The DB2 ACS API driver must support the use of the following wildcards in the query:

- `DB2ACS_WILDCARD` in string fields
- `DB2ACS_ANY_PARTITIONNUM` for database partition fields
- `DB2ACS_ANY_UINT32` for 32-bit unsigned integer (Uint32) fields

pControlBlock
Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSBeginQuery(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 16. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

db2ACSBeginQuery() does not return any query data.

db2ACSGetNextObject - list next snapshot backup object available to use for restore

Returns the next item in a list of snapshot backup objects that are available to be used for a restore operation.

Include file

`db2ACSApi.h`

Syntax and data structures

```
db2ACS_RC db2ACSGetNextObject(
              db2ACS_QueryOutput      * pQueryOutput,
              db2ACS_CB               * pControlBlock,
              db2ACS_ReturnCode       * pRC );
```

Parameters

pQueryOutput
Data type: `db2ACS_QueryOutput *`

`db2ACS_QueryOutput` contains query result information about snapshot backup objects.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pQueryOutput** before returning.

pControlBlock
Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSGetNextObject(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 17. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_OBJ_NOT_FOUND`	The DB2 ACS API driver could not find the snapshot backup object specified by the database manager.	The function call didn't fail, but there are no snapshot backup objects that match the criteria passed to db2ACSBeginQuery().
`DB2ACS_RC_END_OF_DATA`	The DB2 ACS API driver cannot find any more snapshot backup objects.	The function call didn't fail, but there are no more snapshot backup objects that match the criteria passed to db2ACSBeginQuery().
`DB2ACS_RC_MORE_DATA`	There is more data to be transferred from the storage location to the database manager.	Information about a snapshot backup object that matches the criteria passed to db2ACSBeginQuery() is returned, and there are more snapshot backup objects that that match the criteria passed to db2ACSBeginQuery().

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

The database manager must call db2ACSBeginQuery() before calling db2ACSGetNextObject(). The database manager specifies the search criteria in the `db2ACS_QueryInput` parameter passed to db2ACSBeginQuery().

db2ACSGetNextObject() returns information about one snapshot backup object that matches the search criteria passed to db2ACSBeginQuery(). If db2ACSGetNextObject() returns `DB2ACS_RC_MORE_DATA`, the database manager can call db2ACSGetNextObject() again to receive information about another snapshot backup object that matches the search criteria. If db2ACSGetNextObject() returns `DB2ACS_RC_END_OF_DATA`, there are no more snapshot backup objects that match the search criteria.

db2ACSEndQuery - end a query about snapshot backup objects

The database manager uses the DB2 Advanced Copy Services (ACS) API functions db2ACSBeginQuery() and db2ACSGetNextObject() to query about snapshot backup objects that are available to use for restore operations. db2ACSEndQuery() terminates that DB2 ACS query session.

Include file

`db2ACSApi.h`

Syntax and data structures

```
db2ACS_RC db2ACSEndQuery(
             db2ACS_CB              * pControlBlock,
             db2ACS_ReturnCode      * pRC );
```

Parameters

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSEndQuery(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 18. Return codes

Return code	Description	Notes
DB2ACS_RC_OK	The operation was successful.	
DB2ACS_RC_INV_ACTION	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_INV_DEV_HANDLE	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_DEV_ERROR	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_IO_ERROR	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

The database manager cannot call db2ACSGetNextObject() again on this DB2 ACS session without first calling db2ACSBeginQuery() again.

db2ACSSnapshot - perform a DB2 Advanced Copy Services (ACS) operation

Performs a DB2 Advanced Copy Services (ACS) operation.

Include file

`db2ACSApi.h`

Syntax and data structures

```
typedef union db2ACS_ReadList
{
   db2ACS_GroupList           group;
} db2ACS_ReadList;


db2ACS_RC db2ACSSnapshot(
               db2ACS_Action             action,
               db2ACS_ObjectID           objectID,
        db2ACS_ReadList         * pReadList,
        db2ACS_CB               * pControlBlock,
        db2ACS_ReturnCode       * pRC );
```

Parameters

action Data type: `db2ACS_Action`

The type of DB2 ACS action to perform. Values:

DB2ACS_ACTION_WRITE
DB2ACS_ACTION_READ_BY_OBJECT
DB2ACS_ACTION_READ_BY_GROUP

The database manager passes **action** in to db2ACSSnapshot().

objectID
Data type: `db2ACS_ObjectID`

A `db2ACS_ObjectID` is a unique identifier for each stored object, which is returned by a query to the storage repository. A `db2ACS_ObjectID` is guaranteed to be unique and persistent only within the the timeframe of a single DB2 ACS session.

If the database manager specified `DB2ACS_OP_READ` or `DB2ACS_OP_DELETE` as **operation** in the call to db2ACSBeginOperation(), then the database manager passes the value for **objectID** in to db2ACSSnapshot().

pReadList
Data type: `db2ACS_ReadList *`

`db2ACS_ReadList` contains a list of groups.

pReadList is only used if **action** is `DB2ACS_ACTION_READ_BY_GROUP`.

If **action** is `DB2ACS_ACTION_READ_BY_GROUP`, then the database manager is responsible for allocating memory for and populating the fields of **pReadLIst** before calling db2ACSSnapshot(), and for freeing the memory for **pReadList** afterwards.

pControlBlock
Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSSnapshot(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 19. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:
- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

The database manager calls db2ACSBeginOperation() before calling db2ACSPartition(), db2ACSPrepare(), and then db2ACSSnapshot(). The database manager specifies the type of DB2 ACS operation that the DB2 ACS API driver should perform in the **operation** parameter in the call to db2ACSBeginOperation().

db2ACSPartition - group target data for a database partition together

Associates a group identifier with each of the paths listed by the database manager as belonging to a database partition.

Include file

`db2ACSApi.h`

Syntax and data structures

```
/* ==========================================================================
 * Partition
 * ========================================================================== */
db2ACS_RC db2ACSPartition(
             db2ACS_PathList         * pPathList,
       db2ACS_CreateObjectInfo * pCreateObjInfo,
       db2ACS_CB               * PControlBlock,
       db2ACS_ReturnCode       * pRC );
```

Parameters

pPathList

Data type: `db2ACS_PathList`

`db2ACS_PathList` contains a list of database paths, including some extra information about each of those paths specific to DB2 ACS operations.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The **entry** field of the `db2ACS_PathList` structure is an array of elements of type `db2ACS_PathEntry`. `db2ACS_PathEntry` contains information about a database path.

Before calling db2ACSPartition, the database manager populates the following fields of each `db2ACS_PathEntry` entry in **pPathList**:

- **path**
- **type**
- **toBeExcluded**

Every path identified by the database manager as belonging to this database partition is given a group identifier by the DB2 ACS API driver. The DB2 ACS API driver populates the **groupID** field of each `db2ACS_PathEntry` in **pPathList** before returning.

pCreateObjInfo

Data type: `db2ACS_CreateObjectInfo`

`db2ACS_CreateObjectInfo` contains information about the DB2 ACS backup object creation.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The database manager populates the fields of **pCreateObjInfo** before calling db2ACSPartition.

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSPartition(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC

Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 20. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INIT_FAILED`	The database manager attempted to initialize a DB2 ACS session, but the initialization failed.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_OBJ_OUT_OF_SCOPE`	The database manager attempted to perform a DB2 ACS operation on a recovery object that is not managed by the DB2 ACS API driver.	

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: “DB2 Advanced Copy Services (ACS) API return codes” on page 336.

Usage notes

DB2 Advanced Copy Services handles the data on a single database partition atomically. That is: the data for one database partition is backed up or restored together, and independently of other database partitions - even when the action is part of an operation involving multiple database partitions. db2ACSPartition groups database path information for a single database partition together.

The database manager calls db2ACSPartition before calling db2ACSSnapshot. The database manager will list all the paths associated with this database partition in the **pPathList** parameter. The database manager can perform a DB2 ACS operation on a subset of the paths listed in **pPathList** by specifying that subset of paths in the **pReadList** parameter passed to db2ACSSnapshot.

db2ACSVerify - verify that a DB2 Advanced Copy Services (ACS) operation has completed successfully

Verifies that a DB2 Advanced Copy Services (ACS) operation succeeded

Include file

`db2ACSApi.h`

Syntax and data structures

```
/* ==========================================================================
 * Verify
 * ========================================================================== */
db2ACS_RC db2ACSVerify(
             db2ACS_PostObjectInfo   * pPostObjInfo,
             db2ACS_CB               * pControlBlock,
             db2ACS_ReturnCode       * pRC );
```

Parameters

pPostObjInfo

Data type: `db2ACS_PostObjectInfo`

`db2ACS_DB2ID` is a set of data that can not be known at snapshot backup object creation time, but which must be maintained in the object repository.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The database manager populates the fields of **pPostObjInfo** before calling db2ACSVerify. **pPostObjInfo** contains information that is relevant after the DB2 ACS operation. For example, after a successful snapshot backup, **pPostObjInfo** might contain the first active log file. If there is no data relevant for after the DB2 ACS operation, then the database manager will set **pPostObjInfo** to `NULL`.

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSVerify(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 21. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:
- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

If db2ACSVerify returns that a snapshot backup operation succeeded, that means that the recovery objects generated by the snapshot backup are available to be used for restore operations.

db2ACSDelete - delete recovery objects that were created using DB2 Advanced Copy Services (ACS)

Deletes recovery objects that were created using DB2 Advanced Copy Services (ACS)

Include file

`db2ACSApi.h`

Syntax and data structures

```
/* ==========================================================================
 * Delete
 * ========================================================================== */
db2ACS_RC db2ACSDelete(
                    db2ACS_ObjectID          objectID,
                    db2ACS_CB              * pControlBlock,
                    db2ACS_ReturnCode      * pRC );
```

Parameters

objectID
Data type: `db2ACS_ObjectID`

A `db2ACS_ObjectID` is a unique identifier for each stored object, which is returned by a query to the storage repository. A `db2ACS_ObjectID` is guaranteed to be unique and persistent only within the the timeframe of a single DB2 ACS session.

The database manager can use db2ACSQuery() to obtain a valid **objectID** to pass to db2ACSDelete().

pControlBlock
Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSDelete(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 22. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	The specified object is deleted. No further DB2 ACS operations can be performed on that object.
`DB2ACS_RC_DELETE_FAILED`	The DB2 ACS API driver could not successfully delete snapshot backup objects specified by the database manager.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

Table 22. Return codes (continued)

Return code	Description	Notes
DB2ACS_RC_IO_ERROR	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_OBJ_NOT_FOUND	The DB2 ACS API driver could not find the snapshot backup object specified by the database manager.	

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

When the database manager calls db2ACSDelete, the DB2 ACS API driver deletes the recovery object identified by **objectID**.

The database manager calls db2ACSDelete when a user calls db2acsutil with the DELETE parameter.

db2ACSStoreMetaData - store metadata for a recovery object generated using DB2 Advanced Copy Services (ACS)

Stores metadata about a recovery object that was created using DB2 Advanced Copy Services (ACS)

Include file

`db2ACSApi.h`

Syntax and data structures

```
db2ACS_RC db2ACSStoreMetaData(
             db2ACS_MetaData         * pMetaData,
             db2ACS_CB               * pControlBlock,
             db2ACS_ReturnCode       * pRC );
```

Parameters

pMetaData
Data type: `db2ACS_MetaData`

`db2ACS_MetaData` stores snapshot backup meta data.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The metadata stored in the **data** field of **pMetaData** is internal to the database manager, and might change over time, so the DB2 ACS API driver just treats this data as a binary stream.

pControlBlock
Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSStoreMetaData(), the database manager populates the following fields:

```
pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options
```

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 23. Return codes

Return code	Description	Notes
DB2ACS_RC_OK	The operation was successful.	
DB2ACS_RC_INV_ACTION	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_INV_DEV_HANDLE	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_DEV_ERROR	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
DB2ACS_RC_IO_ERROR	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: "DB2 Advanced Copy Services (ACS) API return codes" on page 336.

Usage notes

A snapshot backup operation is comprised of several DB2 ACS API function calls such as: db2ACSInitialize, db2ACSBeginOperation, db2ACSPrepare, and db2ACSSnapshot. db2ACSStoreMetaData is part of the overall operation too. All of these API calls, including db2ACSStoreMetaData must succeed for the snapshot backup operation to succeed. If db2ACSStoreMetaData fails, the recovery object that was generated by the DB2 ACS backup operation is unusable.

db2ACSRetrieveMetaData - retrieve metadata about a recovery object generated using DB2 Advanced Copy Services (ACS)

Retrieves metadata about a recovery object that was created using DB2 Advanced Copy Services (ACS)

Include file

`db2ACSApi.h`

Syntax and data structures

```
db2ACS_RC db2ACSRetrieveMetaData(
              db2ACS_MetaData         * pMetaData,
              db2ACS_ObjectID           objectID,
              db2ACS_CB               * pControlBlock,
              db2ACS_ReturnCode       * pRC );
```

Parameters

pMetaData

Data type: `db2ACS_MetaData`

`db2ACS_MetaData` stores snapshot backup meta data.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The metadata stored in the **data** field of **pMetaData** is internal to the database manager, and might change over time, so the DB2 ACS API driver just treats this data as a binary stream.

objectID

Data type: `db2ACS_ObjectID`

A `db2ACS_ObjectID` is a unique identifier for each stored object, which is returned by a query to the storage repository. A `db2ACS_ObjectID` is guaranteed to be unique and persistent only within the the timeframe of a single DB2 ACS session.

The database manager can use db2ACSQuery() to obtain a valid **objectID** to pass to db2ACSRetrieveMetaData().

pControlBlock

Data type: `db2ACS_CB *`

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

Before calling db2ACSRetrieveMetaData(), the database manager populates the following fields:

pControlBlock->handle
pControlBlock->vendorInfo
pControlBlock->options

pRC Data type: `db2ACS_ReturnCode *`

`db2ACS_ReturnCode` contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a `db2ACS_ReturnCode` parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

The database manager allocates the memory for this parameter, and passes a pointer to that instantiated object to the function. The database manager is responsible for freeing this memory.

The DB2 ACS API driver populates the fields of **pRC** before returning.

Return Codes

Table 24. Return codes

Return code	Description	Notes
`DB2ACS_RC_OK`	The operation was successful.	
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.
`DB2ACS_RC_OBJ_NOT_FOUND`	The DB2 ACS API driver could not find the snapshot backup object specified by the database manager.	The DB2 ACS API driver encountered an error. The database manager cannot use the DB2 ACS API session.

If the DB2 ACS API driver encounters an error, the driver might abort a DB2 ACS operation. The DB2 ACS session cannot be used for any action other than the following:

- If a call to db2ACSBeginQuery() previously succeeded the database manager can call db2ACSEndQuery()
- If a call to db2ACSBeginOperation() previously succeeded, the database manager can call db2ACSEndOperation()
- If a call to db2ACSInitialize() previously succeeded, the database manager can call db2ACSTerminate()

For more information about DB2 ACS API return codes, see the topic: “DB2 Advanced Copy Services (ACS) API return codes” on page 336.

Usage notes

None.

DB2 Advanced Copy Services (ACS) API data structures

To call DB2 Advanced Copy Services (ACS) API functions, you must use DB2 ACS API data structures.

db2ACS_BackupDetails DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_BackupDetails` contains information about a snapshot backup operation.

```
/* ------------------------------------------------------------------------- */
typedef struct db2ACS_BackupDetails
{
   /* A traditional DB2 backup can consist of multiple objects (logical tapes),
    * where each object is uniquely numbered with a non-zero natural number.
    * ---------------------------------------------------------------------- */
   db2Uint32                sequenceNum;

   char                     imageTimestamp[SQLU_TIME_STAMP_LEN + 1];
} db2ACS_BackupDetails;
```

sequenceNum

Data type: `db2Uint32`.

Identifies a backup object by its unique number.

imageTimestamp

Data type: `char[]`.

A character string of length `SQLU_TIME_STAMP_LEN` + 1.

db2ACS_CB DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_CB` contains fundamental information required to initialize and terminate a DB2 ACS session.

```
/* =========================================================================
 * DB2 Backup Adapter Control Block
 * ========================================================================= */
typedef struct db2ACS_CB
{
   /* Output:  Handle value for this session.
    * ---------------------------------------------------------------------- */
   db2Uint32                handle;
   db2ACS_VendorInfo        vendorInfo;

   /* Input fields and parameters.
    * ---------------------------------------------------------------------- */
   db2ACS_SessionInfo       session;
   db2ACS_Options           options;

   /* Operation info is optional, possibly NULL, and is only ever valid
    * within the context of an operation (from call to BeginOperation() until
    * the EndOperation() call returns).
    *
    * The operation info will be present during creation or read operations
    * of snapshot and backup objects.
    * ---------------------------------------------------------------------- */
   db2ACS_OperationInfo   * operation;
} db2ACS_CB;
```

handle

Data type: `db2Uint32`.

A handle to reference the DB2 ACS session.

vendorInfo

Data type: `db2ACS_VendorInfo`.

db2ACS_VendorInfo contains information about the DB2 ACS API driver.

session
Data type: db2ACS_SessionInfo.

db2ACS_SessionInfo contains all the information about the DB2 ACS session.

options
Data type: db2ACS_Options.

db2ACS_Options specifies options to be used for a DB2 ACS operation. This contents of this string is specific to the DB2 ACS API driver.

operation
Data type: db2ACS_OperationInfo *.

db2ACS_OperationInfo contains information about a snapshot backup operation.

db2ACS_CreateObjectInfo DB2 Advanced Copy Services (ACS) API data structure

db2ACS_CreateObjectInfo contains information about the DB2 ACS backup object creation.

```
/* ==========================================================================
 * Object Creation Parameters.
 * ========================================================================== */
typedef struct db2ACS_CreateObjectInfo
{
   db2ACS_ObjectInfo          object;
   db2ACS_DB2ID               db2ID;

   /* -----------------------------------------------------------------------
    * The following fields are optional information for the database manager
    * to use as it sees fit.
    * ----------------------------------------------------------------------- */

   /* Historically both the size estimate and management
    * class parameters have been used by the TSM client API for traditional
    * backup objects, log archives, and load copies, but not for snapshot
    * backups.
    * ----------------------------------------------------------------------- */
   db2Uint64                  sizeEstimate;
   char                       mgmtClass[DB2ACS_MAX_MGMTCLASS_SZ + 1];

   /* The appOptions is a copy of the iOptions field of flags passed to DB2's
    * db2Backup() API when this execution was initiated.  This field will
    * only contain valid data when creating a backup or snapshot object.
    * ----------------------------------------------------------------------- */
   db2Uint32                  appOptions;
} db2ACS_CreateObjectInfo;
```

object Data type: db2ACS_ObjectInfo

db2ACS_ObjectInfo contains information about object created using the DB2 Advanced Copy Services (ACS) API.

db2ID Data type: db2ACS_DB2ID

db2ACS_DB2ID identifies the IBM Data Server.

sizeEstimate
Data type: db2Uint64.

An estimate of the size of backup objects being created. This estimate does not apply to log archives, load copies, or snapshot backups objects.

mgmtClass
Data type: db2ACS_MgmtClass.

A character string of length db2ACS_MAX_MGMTCLASS_SZ + 1.

This does not apply to snapshot backup objects.

appOptions
Data type: db2Uint32.

A copy of the backup options passed to the backup command that initiated the snapshot backup.

db2ACS_DB2ID DB2 Advanced Copy Services (ACS) API data structure

db2ACS_DB2ID identifies the IBM Data Server.

```
/* ==========================================================================
 * DB2 Data Server Identifier
 * ========================================================================== */
typedef struct db2ACS_DB2ID
{
   db2Uint32                  version;
   db2Uint32                  release;
   db2Uint32                  level;
   char                       signature[DB2ACS_SIGNATURE_SZ + 1];
} db2ACS_DB2ID;
```

version
Data type: db2Uint32.

Version of IBM Data Server. For example: 9

release
Data type: db2Uint32.

Release level of IBM Data Server. For example: 5

level Data type: db2Uint32.

Level identifier for the IBM Data Server. For example: 0

signature
Data type: char[].

A character string of length DB2ACS_SIGNATURE_SZ + 1. For example: ″SQL09050″

db2ACS_GroupList DB2 Advanced Copy Services (ACS) API data structure

db2ACS_GroupList contains a list of groups to be included in the snapshot backup operation.

```
/* ==========================================================================
 * Snapshot Group List
 *
 * This is an array of size 'numGroupIDs', indicating the set of groups that
 * are to be included in the snapshot operation.
 * ========================================================================== */
typedef struct db2ACS_GroupList
{
   db2Uint32                  numGroupIDs;
   db2Uint32                * id;
} db2ACS_GroupList;
```

numGroupIDs
Data type: db2Uint32.

Number of groups in the array **id**.

id Data type: `db2Uint32 *`.

An array of group identifiers. The groups identified are the groups (or lists of paths) to be included in the snapshot backup operation.

db2ACS_LoadcopyDetails DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_LoadcopyDetails` contains information about a load copy operation.

```
/* ------------------------------------------------------------------------ */
typedef struct db2ACS_LoadcopyDetails
{
   /* Just like the BackupDetails, a DB2 load copy can consist of multiple
    * objects (logical tapes), where each object is uniquely numbered with a
    * non-zero natural number.
    * --------------------------------------------------------------------- */
   db2Uint32                 sequenceNum;

   char                      imageTimestamp[SQLU_TIME_STAMP_LEN + 1];
} db2ACS_LoadcopyDetails;
```

sequenceNum
Data type: `db2Uint32`.

Identifies a backup object by its unique number.

imageTimestamp
Data type: `char[]`.

A character string of length `SQLU_TIME_STAMP_LEN + 1`

db2ACS_LogDetails DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_LogDetails` contains information that identifies a particular database log file.

```
/* ------------------------------------------------------------------------ */
typedef struct db2ACS_LogDetails
{
   db2Uint32                 fileID;
   db2Uint32                 chainID;
} db2ACS_LogDetails;
```

fileID Data type: `db2Uint32`.

A number which is the file name of the database log file.

chainID
Data type: `db2Uint32`.

A number which identifies the database log file chain to which the database log file **fileID** belongs.

db2ACS_ObjectInfo DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_ObjectInfo` contains information about object created using the DB2 Advanced Copy Services (ACS) API.

```
/* =========================================================================
 * Object Description and Associated Information.
 *
 * This structure is used for both input and output, and its contents define
 * the minimum information that must be recorded about any object created
 * through this interface.
 * ====================================================================== */
```

```
typedef struct db2ACS_ObjectInfo
{
   db2ACS_ObjectType          type;
   SQL_PDB_NODE_TYPE          dbPartitionNum;

   char                       db[SQL_DBNAME_SZ + 1];
   char                       instance[DB2ACS_MAX_OWNER_SZ + 1];
   char                       host[SQL_HOSTNAME_SZ + 1];
   char                       owner[DB2ACS_MAX_OWNER_SZ + 1];

   union
   {
      db2ACS_BackupDetails    backup;
      db2ACS_LogDetails       log;
      db2ACS_LoadcopyDetails  loadcopy;
      db2ACS_SnapshotDetails  snapshot;
   } details;
} db2ACS_ObjectInfo;
```

type Data type: `db2ACS_ObjectType`.

Specifies the snapshot backup objects type. Values:

DB2ACS_OBJTYPE_ALL
DB2ACS_OBJTYPE_BACKUP
DB2ACS_OBJTYPE_LOG
DB2ACS_OBJTYPE_LOADCOPY
DB2ACS_OBJTYPE_SNAPSHOT

DB2ACS_OBJTYPE_ALL can only be used as a filter for queries. There are no objects of type 0.

dbPartitionNum
Data type: `SQL_PDB_NODE_TYPE`.

An identifier for this database partition.

db Data type: `char[]`.

A character string of length `SQL_DBNAME_SZ` + 1.

instance
Data type: `char[]`.

A character string of length `DB2ACS_MAX_OWNER_SZ` + 1.

host Data type: `char[]`.

A character string of length `SQL_HOSTNAME_SZ` + 1.

owner Data type: `char[]`.

A character string of length `DB2ACS_MAX_OWNER_SZ` + 1.

details

backup
Data type: `db2ACS_BackupDetails`

`db2ACS_BackupDetails` contains information about a snapshot backup operation.

log Data type: `db2ACS_LogDetails`

`db2ACS_LogDetails` contains information that identifies a particular database log file.

loadcopy
Data type: `db2ACS_LoadcopyDetails`

`db2ACS_LoadcopyDetails` contains information about a load copy operation.

snapshot
Data type: `db2ACS_SnapshotDetails`

`db2ACS_SnapshotDetails` contains information about a snapshot backup operation.

db2ACS_ObjectStatus DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_ObjectStatus` contains information about the status or progress of a snapshot backup operation, or the status or usability of a snapshot backup object.

```
typedef struct db2ACS_ObjectStatus
{
   /* The total and completed bytes refer only to the ACS snapshot backup
    * itself, not to the progress of any offloaded tape backup.
    *
    * A bytesTotal of 0 indicates that the progress could not be determined.
    * ---------------------------------------------------------------------- */
   db2Uint64                bytesCompleted;
   db2Uint64                bytesTotal;
   db2ACS_ProgressState     progressState;
   db2ACS_UsabilityState    usabilityState;
} db2ACS_ObjectStatus;
```

bytesCompleted
Data type: `db2Uint64`.

The amount of the snapshot backup that has completed, in bytes.

bytesTotal
Data type: `db2Uint64`.

The size of the completed snapshot backup, in bytes.

progressState
Data type: `db2ACS_ProgressState`.

The state of the snapshot backup operation. Values:

DB2ACS_PSTATE_UNKNOWN
DB2ACS_PSTATE_IN_PROGRESS
DB2ACS_PSTATE_SUCCESSFUL
DB2ACS_PSTATE_FAILED

usabilityState
Data type: `db2ACS_UsabilityState`.

The state of the snapshot backup object, how the snapshot backup object can be used. Values:

DB2ACS_USTATE_UNKNOWN
DB2ACS_USTATE_LOCALLY_MOUNTABLE
DB2ACS_USTATE_REMOTELY_MOUNTABLE
DB2ACS_USTATE_REPETITIVELY_RESTORABLE
DB2ACS_USTATE_DESTRUCTIVELY_RESTORABLE
DB2ACS_USTATE_SWAP_RESTORABLE
DB2ACS_USTATE_PHYSICAL_PROTECTION
DB2ACS_USTATE_FULL_COPY

DB2ACS_USTATE_DELETED
DB2ACS_USTATE_FORCED_MOUNT
DB2ACS_USTATE_BACKGROUND_MONITOR_PENDING
DB2ACS_USTATE_TAPE_BACKUP_PENDING
DB2ACS_USTATE_TAPE_BACKUP_IN_PROGRESS
DB2ACS_USTATE_TAPE_BACKUP_COMPLETE

db2ACS_OperationInfo DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_OperationInfo` contains information about a snapshot backup operation.

```
/* ==========================================================================
 * Operation Info
 *
 * The information contained within this structure is only valid within the
 * context of a particular operation.  It will be valid at the time
 * BeginOperation() is called, and will remain unchanged until EndOperation()
 * returns, but must not be referenced outside the scope of an operation.
 * ========================================================================== */
typedef struct db2ACS_OperationInfo
{
   db2ACS_SyncMode            syncMode;

   /* List of database and backup operation partitions.
    *
    * For details, refer to the db2ACS_PartitionList definition.
    * ----------------------------------------------------------------------- */
   db2ACS_PartitionList     * dbPartitionList;
} db2ACS_OperationInfo;
```

syncMode

Data type: `db2ACS_SyncMode`.

The level of synchronization between the backup operations on separate database partitions.

Values:

DB2ACS_SYNC_NONE
No synchronization between related operations on multiple database partitions. Used during operations which do not make use of any synchronization between the multiple database partitions.

DB2ACS_SYNC_SERIAL
Used when performing concurrent snapshot backup operations on multiple database partitions. Each database partition will have its input and output (IO) suspended when the snapshot backup operation is issued, and then the IO on the database partitions is resumed serially, not concurrently.

SYNC_PARALLEL
Performing a snapshot operation on multiple partitions concurrently. Once all database partitions that are involved in the snapshot backup operation have completed preparations for the snapshot backup operation, input and output (IO) will be suspended on all of the database partitions. The remaining snapshot backup steps will take place concurrently on all of the involved database partitions.

dbPartitionList

Data type: `db2ACS_PartitionList *`.

db2ACS_PartitionList contains information about the database partitions that are in the database and that are involved in a DB2 ACS operation.

db2ACS_Options DB2 Advanced Copy Services (ACS) API data structure

db2ACS_Options specifies options to be used for a DB2 ACS operation. This contents of this string is specific to the DB2 ACS API driver.

```
/* ==========================================================================
 * DB2 Backup Adapter User Options
 * ========================================================================== */
typedef struct db2ACS_Options
{
   db2Uint32                  size;
   void                     * data;
} db2ACS_Options;
```

size Data type: db2Uint32.

Size of **data**, in bytes.

data Data type: void *.

Pointer to a block of memory that contains the options.

db2ACS_PartitionEntry DB2 Advanced Copy Services (ACS) API data structure

db2ACS_PartitionEntry is an element of a db2ACS_PartitionList.

```
typedef struct db2ACS_PartitionEntry
{
   SQL_PDB_NODE_TYPE          num;
   char                       host[SQL_HOSTNAME_SZ + 1];
} db2ACS_PartitionEntry;
```

num Data type: SQL_PDB_NODE_TYPE.

An identifier for this database partition entry.

host Data type: char[].

A character string of length SQL_HOSTNAME_SZ + 1.

db2ACS_PartitionList DB2 Advanced Copy Services (ACS) API data structure

db2ACS_PartitionList contains information about the database partitions that are in the database and that are involved in a DB2 ACS operation.

```
typedef struct db2ACS_PartitionList
{
   db2Uint64                  numPartsInDB;
   db2Uint64                  numPartsInOperation;
   db2ACS_PartitionEntry    * partition;
} db2ACS_PartitionList;
```

numPartsInDB
Data type: db2Uint64.

The number of database partitions in the database.

numPartsInOperation
Data type: db2Uint64.

The number of database partitions involved in the DB2 ACS operation.

partition
Data type: db2ACS_PartitionEntry *.

db2ACS_PartitionEntry is an element of a db2ACS_PartitionList.

db2ACS_PathEntry DB2 Advanced Copy Services (ACS) API data structure

db2ACS_PathEntry contains information about a database path.

```
typedef struct db2ACS_PathEntry
{
   /* INPUT: The path and type will be provided by the database server, as well
    *       as a flag indicating if the path is to be excluded from the backup.
    * --------------------------------------------------------------------- */
   char                     path[DB2ACS_MAX_PATH_SZ + 1];
   db2ACS_PathType          type;
   db2Uint32                toBeExcluded;

   /* OUTPUT: The group ID is to be provided by the backup adapter for use by
    *        the DB2 server.  The group ID will be used during with snapshot
    *        operations as an indication of which paths are dependent and must
    *        be included together in any snapshot operation.  Unique group IDs
    *        indicate that the paths in those groups are independent for the
    *        purposes of snapshot operations.
    * --------------------------------------------------------------------- */
   db2Uint32                groupID;
} db2ACS_PathEntry;
```

path Data type: char[].

A character string of length DB2ACS_MAX_PATH_SZ + 1.

type Data type: db2ACS_PathType.

The type of path. Values:

DB2ACS_PATH_TYPE_UNKNOWN
DB2ACS_PATH_TYPE_LOCAL_DB_DIRECTORY
DB2ACS_PATH_TYPE_DBPATH
DB2ACS_PATH_TYPE_DB_STORAGE_PATH
DB2ACS_PATH_TYPE_TBSP_CONTAINER
DB2ACS_PATH_TYPE_TBSP_DIRECTORY
DB2ACS_PATH_TYPE_TBSP_DEVICE
DB2ACS_PATH_TYPE_LOGPATH
DB2ACS_PATH_TYPE_MIRRORLOGPATH

toBeExcluded
Data type: db2Uint32.

A flag indicating whether to include the given path in the snapshot backup. Values:

- 0 - include the path in the snapshot backup
- 1 - do not include the path in the snapshot backup

groupID
Data type: db2Uint32.

A group identifier.

db2ACS_PathList DB2 Advanced Copy Services (ACS) API data structure

db2ACS_PathList contains a list of database paths, including some extra information about each of those paths specific to DB2 ACS operations.

```
/* ==========================================================================
 * Snapshot File List
 *
 * This is an array of 'numEntries' db2ACS_PathEntry's, where each path entry is
 * a path to some storage on the DB2 server which is in use by the current
 * database.
 * ========================================================================== */

typedef struct db2ACS_PathList
{
   db2Uint32                 numEntries;
   db2ACS_PathEntry        * entry;
} db2ACS_PathList;
```

numEntries
Data type: db2Uint32.

The number of path entries in the **entry** array.

entry Data type: db2ACS_PathEntry.

db2ACS_PathEntry contains information about a database path.

db2ACS_PostObjectInfo DB2 Advanced Copy Services (ACS) API data structure

db2ACS_DB2ID is a set of data that can not be known at snapshot backup object creation time, but which must be maintained in the object repository.

```
/* ==========================================================================
 * The PostObjectInfo is a set of data that can not be known at object
 * creation time, but which must be maintained in the object repository.  This
 * is an optional field on the Verify() call, which may be NULL if there are
 * no post-operation updates to be made.
 * ========================================================================== */
typedef struct db2ACS_PostObjectInfo
{
   /* The first active log will only be valid when creating a backup or
    * snapshot object.  It will indicate the file number and chain id of the
    * first log required for recovery using this object.
    * ----------------------------------------------------------------------- */
   db2ACS_LogDetails          firstActiveLog;
} db2ACS_PostObjectInfo;
```

firstActiveLog
Data type: db2ACS_LogDetails.

db2ACS_LogDetails contains information that identifies a particular database log file.

db2ACS_QueryInput and db2ACS_QueryOutput DB2 Advanced Copy Services (ACS) API data structures

db2ACS_QueryInput contains identifying information for an object about which you are querying. db2ACS_QueryOutput contains query result information about snapshot backup objects.

```
/* ==========================================================================
 * Unique Querying.
 *
 * When using this structure as query input, to indicate the
 * intention to supply a 'wildcard' search criteria, DB2 will supply:
 *
 *    -- character strings as "*".
 *    -- numeric values as (-1), cast as the appropriate signed or unsigned
 *       type.
 * ========================================================================== */

typedef struct db2ACS_ObjectInfo db2ACS_QueryInput;

typedef struct db2ACS_QueryOutput
{
   db2ACS_ObjectID            objectID;
   db2ACS_ObjectInfo          object;
   db2ACS_PostObjectInfo      postInfo;
   db2ACS_DB2ID               db2ID;
   db2ACS_ObjectStatus        status;

   /* Size of the object in bytes.
    * --------------------------------------------------------------------  */
   db2Uint64                  objectSize;

   /* Size of the metadata associated with the object, if any, in bytes.
    * --------------------------------------------------------------------  */
   db2Uint64                  metaDataSize;

   /* The creation time of the object is a 64bit value with a definition
    * equivalent to an ANSI C time_t value (seconds since the epoch, GMT).
    *
    * This field is equivalent to the file creation or modification time in
    * a traditional filesystem.  This should be created and stored
    * automatically by the BA subsystem, and a valid time value should be
    * returned with object query results, for all object types.
    * --------------------------------------------------------------------  */
   db2Uint64                  createTime;
} db2ACS_QueryOutput;
```

objectID
Data type: db2ACS_ObjectID.

A db2ACS_ObjectID is a unique identifier for each stored object, which is returned by a query to the storage repository. A db2ACS_ObjectID is guaranteed to be unique and persistent only within the the timeframe of a single DB2 ACS session.

object Data type: db2ACS_ObjectInfo

db2ACS_ObjectInfo contains information about object created using the DB2 Advanced Copy Services (ACS) API.

postInfo
Data type: db2ACS_PostObjectInfo.

db2ACS_DB2ID is a set of data that can not be known at snapshot backup object creation time, but which must be maintained in the object repository.

db2ID Data type: db2ACS_DB2ID.

db2ACS_DB2ID identifies the IBM Data Server.

status Data type: db2ACS_ObjectStatus.

db2ACS_ObjectStatus contains information about the status or progress of a snapshot backup operation, or the status or usability of a snapshot backup object.

objectSize
Data type: db2Uint64.

Size of the object in bytes.

metaDataSize
Data type: db2Uint64.

Size of the metadata associated with the object, if any, in bytes.

createTime
Data type: db2Uint64.

The creation time of an object. The value of **createTime** is equivalent to an ANSI C time_t value.

db2ACS_ReadList DB2 Advanced Copy Services (ACS) API data structure

db2ACS_ReadList contains a list of groups.

```
/* The ReadList will only be used for snapshots where the action is READ, and
 * where one of the granularity modifiers other than BY_OBJ has been specified.
 * In the typical usage scenario of ( READ | BY_OBJ ) the ReadList parameter
 * should be ignored.
 *
 * When the action is DB2ACS_ACTION_BY_GROUP the union is to be interpreted
 * as a group list.
 * ------------------------------------------------------------------------- */
typedef union db2ACS_ReadList
{
   db2ACS_GroupList           group;
} db2ACS_ReadList;
```

group Data type: db2ACS_GroupList.

db2ACS_GroupList contains a list of groups to be included in the snapshot backup operation.

db2ACS_ReturnCode DB2 Advanced Copy Services (ACS) API data structure

db2ACS_ReturnCode contains diagnostic information including message text and error codes specific to the storage hardware. The contents of a db2ACS_ReturnCode parameter for a DB2 ACS API function call will be recorded in the database manager diagnostic logs.

```
/* ==========================================================================
 * Storage Adapter Return Code and Diagnostic Data.
 *
 * These will be recorded in the DB2 diagnostic logs, but are intended to be
 * internal return and reason codes from the storage layers which can be used
 * in conjunction with the DB2ACS_RC to provide more detailed diagnostic info.
 * ========================================================================== */
typedef struct db2ACS_ReturnCode
{
   int                        returnCode;
   int                        reasonCode;
   char                       description[DB2ACS_MAX_COMMENT_SZ + 1];
} db2ACS_ReturnCode;
```

returnCode
Data type: int.

Return code specific to the storage hardware.

reasonCode
Data type: int.

Reason code specific to the storage hardware.

description
Data type: char[].

A character string of length DB2ACS_MAX_COMMENT_SZ + 1.

db2ACS_SessionInfo DB2 Advanced Copy Services (ACS) API data structure

db2ACS_SessionInfo contains all the information about the DB2 ACS session.

```
/* ==========================================================================
 * Session Info
 * ========================================================================== */
typedef struct db2ACS_SessionInfo
{
   db2ACS_DB2ID              db2ID;

   /* Fields identifying the backup session originator.
    * ------------------------------------------------------------------- */
   SQL_PDB_NODE_TYPE         dbPartitionNum;
   char                      db[SQL_DBNAME_SZ + 1];
   char                      instance[DB2ACS_MAX_OWNER_SZ + 1];
   char                      host[SQL_HOSTNAME_SZ + 1];
   char                      user[DB2ACS_MAX_OWNER_SZ + 1];
   char                      password[DB2ACS_MAX_PASSWORD_SZ + 1];

   /* The fully qualified ACS vendor library name to be used.
    * ------------------------------------------------------------------- */
   char                      libraryName[DB2ACS_MAX_PATH_SZ + 1];
} db2ACS_SessionInfo;
```

db2ID Data type: `db2ACS_DB2ID`

`db2ACS_DB2ID` identifies the IBM Data Server.

dbPartitionNum
Data type: `SQL_PDB_NODE_TYPE`

The unique, numeric identifier for a database partition.

db Data type: `char[]`.

A character string of length `SQL_DBNAME_SZ` + 1.

instance
Data type: `char[]`.

A character string of length `DB2ACS_MAX_OWNER_SZ` + 1.

host Data type: `char[]`.

A character string of length `SQL_HOSTNAME_SZ` + 1.

user Data type: `char[]`.

A character string of length `DB2ACS_MAX_OWNER_SZ` + 1.

password
Data type: `char[]`.

A character string of length `DB2ACS_MAX_PASSWORD_SZ` + 1.

libraryName
Data type: `char[]`.

A character string of length `DB2ACS_MAX_PATH_SZ` + 1.

db2ACS_SnapshotDetails DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_SnapshotDetails` contains information about a snapshot backup operation.

```
typedef struct db2ACS_SnapshotDetails
{
   char                      imageTimestamp[SQLU_TIME_STAMP_LEN + 1];
} db2ACS_SnapshotDetails;
```

imageTimestamp
Data type: `char[]`.

A character string of length `SQLU_TIME_STAMP_LEN` + 1.

db2ACS_MetaData DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_MetaData` stores snapshot backup meta data.

```
/* ==========================================================================
 * The metadata structure itself is internal to DB2 and is to be treated by
 * the storage interface as an unstructured block of data of the given size.
 * ========================================================================== */
typedef struct db2ACS_MetaData
{
   db2Uint64                 size;
   void                    * data;
} db2ACS_MetaData;
```

size Data type: `db2Uint32`.

Size of **data**, in bytes.

data Data type: `void *`.

A pointer to a block of memory that the database manager uses to store snapshot backup metadata.

db2ACS_VendorInfo DB2 Advanced Copy Services (ACS) API data structure

`db2ACS_VendorInfo` contains information about the DB2 ACS API driver.

```
/* ==========================================================================
 * Storage Vendor Identifier
 * ========================================================================== */
typedef struct db2ACS_VendorInfo
{
   void                    * vendorCB;             /* Vendor control block  */
   db2Uint32                 version;              /* Current version       */
   db2Uint32                 release;              /* Current release       */
   db2Uint32                 level;                /* Current level         */
   char                      signature[DB2ACS_MAX_VENDORID_SZ + 1];
} db2ACS_VendorInfo;
```

vendorCB
Data type: `void *`.

Pointer to a control block that is specific to the DB2 ACS API driver.

version
Data type: `db2Uint32`.

Version of the DB2 ACS API driver.

release
Data type: `db2Uint32`.

Release level of the DB2 ACS API driver.

level Data type: `db2Uint32`.

Level identifier for the DB2 ACS API driver.

signature
Data type: `db2ACS_VendorSignature`.

A character string of length `DB2ACS_MAX_VENDORID_SZ + 1`.

DB2 Advanced Copy Services (ACS) best practices

Consider the following best practices when installing and configuring DB2 Advanced Copy Services (ACS).

Specify a dedicated volume group for log paths
It is recommended that the log paths be contained within their own snapshot volume independent from the database directory and database containers.

Specify one volume group for each database partition
In a database partitioning feature (DPF) environment, each database partition must reside on a set of snapshot volumes independent of the other database partitions.

DB2 Advanced Copy Services (ACS) restrictions

When installing and configuring DB2 Advanced Copy Services (ACS), there are some restrictions to consider.

Volume sharing is not supported. If a database partition resides on the same storage volume as any other database partition, snapshot operations are not permitted.

DB2 Advanced Copy Services (ACS) API return codes

DB2 Advanced Copy Services (ACS) API functions return a defined set of possible return codes.

Table 25. DB2 Advanced Copy Services (ACS) API return codes

Return code	Description
`DB2ACS_RC_OK`	The operation was successful.
`DB2ACS_RC_LINK_EXIST`	The session was previously activated.
`DB2ACS_RC_COMM_ERROR`	There was a communication error with a storage device, such as a tape drive.
`DB2ACS_RC_INV_VERSION`	The version of the database manager's DB2 ACS library and the version of the DB2 ACS API driver are not compatible.
`DB2ACS_RC_INV_ACTION`	The database manager requested an action from the DB2 ACS API driver that is invalid.
`DB2ACS_RC_NO_DEV_AVAIL`	There is currently no storage device, such as a tape drive, available to use.
`DB2ACS_RC_OBJ_NOT_FOUND`	The DB2 ACS API driver could not find the snapshot backup object specified by the database manager.
`DB2ACS_RC_OBJS_FOUND`	The DB2 ACS API driver found more than one snapshot backup object that matches the specification given by the database manager.
`DB2ACS_RC_INV_USERID`	The database manager passed an invalid user id to the DB2 ACS API driver.
`DB2ACS_RC_INV_PASSWORD`	The database manager passed an invalid password to the DB2 ACS API driver.
`DB2ACS_RC_INV_OPTIONS`	The database manager specified invalid options.
`DB2ACS_RC_INIT_FAILED`	The database manager attempted to initialize a DB2 ACS session, but the initialization failed.
`DB2ACS_RC_INV_DEV_HANDLE`	The database manager passed a storage device handle that is invalid.
`DB2ACS_RC_BUFF_SIZE`	The database manager specified a buffer size that is invalid.
`DB2ACS_RC_END_OF_DATA`	The DB2 ACS API driver cannot find any more snapshot backup objects.
`DB2ACS_RC_END_OF_TAPE`	The storage device unexpectedly reached the end of tape backup media.

Table 25. DB2 Advanced Copy Services (ACS) API return codes (continued)

Return code	Description
`DB2ACS_RC_DATA_RESEND`	A storage device, such as a tape drive, requested that the database manager resend the most recent buffer of data.
`DB2ACS_RC_COMMIT_FAILED`	The DB2 ACS API driver could not commit a transaction.
`DB2ACS_RC_DEV_ERROR`	There was an error with a storage device, such as a tape drive.
`DB2ACS_RC_WARNING`	The storage hardware returned a warning. Look in the database manager diagnostic logs for more information.
`DB2ACS_RC_LINK_NOT_EXIST`	The session was not activated previously.
`DB2ACS_RC_MORE_DATA`	There is more data to be transferred from the storage location to the database manager.
`DB2ACS_RC_ENDOFMEDIA_NO_DATA`	The storage device reached the end of the storage media without finding any data.
`DB2ACS_RC_ENDOFMEDIA`	The storage device reached the end of the storage media.
`DB2ACS_RC_MAX_LINK_GRANT`	The maximum number of links has been established. The database manager cannot establish more links.
`DB2ACS_RC_IO_ERROR`	The DB2 ACS API driver encountered an error resulting from input or output operations.
`DB2ACS_RC_DELETE_FAILED`	The DB2 ACS API driver could not successfully delete snapshot backup objects specified by the database manager.
`DB2ACS_RC_INV_BKUP_FNAME`	The database manager specified an invalid filename for the snapshot backup object.
`DB2ACS_RC_NOT_ENOUGH_SPACE`	The DB2 ACS API driver estimated that there is not enough storage space to perform a snapshot backup of the database specified by the database manager.
`DB2ACS_RC_ABORT_FAILED`	The database manager attempted to abort a DB2 ACS operation, but the attempt to abort failed.
`DB2ACS_RC_UNEXPECTED_ERROR`	The DB2 ACS API driver encountered a severe, unknown error.
`DB2ACS_RC_NO_DATA`	The DB2 ACS API driver did not return any data to the database manager.
`DB2ACS_RC_OBJ_OUT_OF_SCOPE`	The database manager attempted to perform a DB2 ACS operation on a recovery object that is not managed by the DB2 ACS API driver.
`DB2ACS_RC_INV_CALL_SEQUENCE`	The database manager made calls to DB2 ACS API functions in a sequence that is invalid. For example, the database manager must call db2ACSInitialize before calling any other DB2 ACS API function except db2ACSQueryAPIVersion.
`DB2ACS_RC_SHARED_STORAGE_GROUP`	The database manager attempted to perform a snapshot operation against a storage object that is being used by another database or application.

DB2 Advanced Copy Services (ACS) supported operating systems and hardware

Integrated into IBM Data Server is a DB2 ACS API driver that supports a subset of the operating systems and hardware that IBM Data Server supports.

Table 26. DB2 Advanced Copy Services (ACS) API supported operating systems and hardware

Hardware	**AIX operating system with storage area network (SAN) storage [2]**	**AIX operating system with network file system (NFS) storage [2]**	**Linux operating system with network file system (NFS) storage [1]**
IBM TotalStorage SAN Volume Controller	Full support.	Not supported.	Not supported.
IBM System Storage DS6000	Full support except: • incremental copying is not supported	Not supported.	Not supported.
IBM System Storage DS8000	Full support except: • incremental copying is not supported	Not supported.	Not supported.
IBM System Storage N Series	Fully supported.	Fully supported.	Fully supported.
NetApp V-series	Fully supported.	Fully supported.	Fully supported.
NetApp FAS series	Fully supported	Fully supported	Fully supported

[1] Only the following systems are supported with DB2 ACS and Linux:

- 64-bit only on x86 (Intel Pentium®, Intel Xeon®, and AMD) processors
- POWER® (IBM eServer™ OpenPower®, System i or pSeries systems that support Linux)

[2] Only AIX 5.3 is supported with DB2 ACS on AIX. For AIX 6.1 operating systems, you can perform a manual copy. An example of this is to suspend write I/O and use the split mirror functionality as described in “Using a split mirror as a backup image” on page 203. You can also purchase and install the full version of Tivoli Storage Manager for Advanced Copy Services V6.1 for AIX 6.1.

Part 3. Appendixes

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/shop/publications/order. 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 27. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC27-2435-00	Yes	August, 2009
Administrative Routines and Views	SC27-2436-00	No	August, 2009
Call Level Interface Guide and Reference, Volume 1	SC27-2437-00	Yes	August, 2009
Call Level Interface Guide and Reference, Volume 2	SC27-2438-00	Yes	August, 2009
Command Reference	SC27-2439-00	Yes	August, 2009
Data Movement Utilities Guide and Reference	SC27-2440-00	Yes	August, 2009
Data Recovery and High Availability Guide and Reference	SC27-2441-00	Yes	August, 2009
Database Administration Concepts and Configuration Reference	SC27-2442-00	Yes	August, 2009
Database Monitoring Guide and Reference	SC27-2458-00	Yes	August, 2009
Database Security Guide	SC27-2443-00	Yes	August, 2009
DB2 Text Search Guide	SC27-2459-00	Yes	August, 2009
Developing ADO.NET and OLE DB Applications	SC27-2444-00	Yes	August, 2009
Developing Embedded SQL Applications	SC27-2445-00	Yes	August, 2009
Developing Java Applications	SC27-2446-00	Yes	August, 2009
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-2447-00	No	August, 2009
Developing User-defined Routines (SQL and External)	SC27-2448-00	Yes	August, 2009
Getting Started with Database Application Development	GI11-9410-00	Yes	August, 2009
Getting Started with DB2 Installation and Administration on Linux and Windows	GI11-9411-00	Yes	August, 2009

Table 27. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Globalization Guide	SC27-2449-00	Yes	August, 2009
Installing DB2 Servers	GC27-2455-00	Yes	August, 2009
Installing IBM Data Server Clients	GC27-2454-00	No	August, 2009
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	SC27-2469-00	No	August, 2009
Partitioning and Clustering Guide	SC27-2453-00	Yes	August, 2009
pureXML Guide	SC27-2465-00	Yes	August, 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	SC27-2468-00	No	August, 2009
SQL Procedural Languages: Application Enablement and Support	SC27-2470-00	Yes	August, 2009
SQL Reference, Volume 1	SC27-2456-00	Yes	August, 2009
SQL Reference, Volume 2	SC27-2457-00	Yes	August, 2009
Troubleshooting and Tuning Database Performance	SC27-2461-00	Yes	August, 2009
Upgrading to DB2 Version 9.7	SC27-2452-00	Yes	August, 2009
Visual Explain Tutorial	SC27-2462-00	No	August, 2009
What's New for DB2 Version 9.7	SC27-2463-00	Yes	August, 2009
Workload Manager Guide and Reference	SC27-2464-00	Yes	August, 2009
XQuery Reference	SC27-2466-00	No	August, 2009

Table 28. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
Installing and Configuring DB2 Connect Personal Edition	SC27-2432-00	Yes	August, 2009
Installing and Configuring DB2 Connect Servers	SC27-2433-00	Yes	August, 2009

Table 28. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print	Last updated
DB2 Connect User's Guide	SC27-2434-00	Yes	August, 2009

Table 29. 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 341.

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 9.7 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 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/

For DB2 Version 8 topics, go to the Version 8 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. Clear the browser cache and then 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. Clear the browser cache and then 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.

Before you begin

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.

About this task

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.

Procedure

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
      ```

Results

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 *DB2 Troubleshooting Guide* 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.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 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

Trademarks

Index

Printed in USA

SC27-2441-00

Spine information:

IBM DB2 9.7 for Linux, UNIX, and Windows

Data Recovery and High Availability Guide and Reference