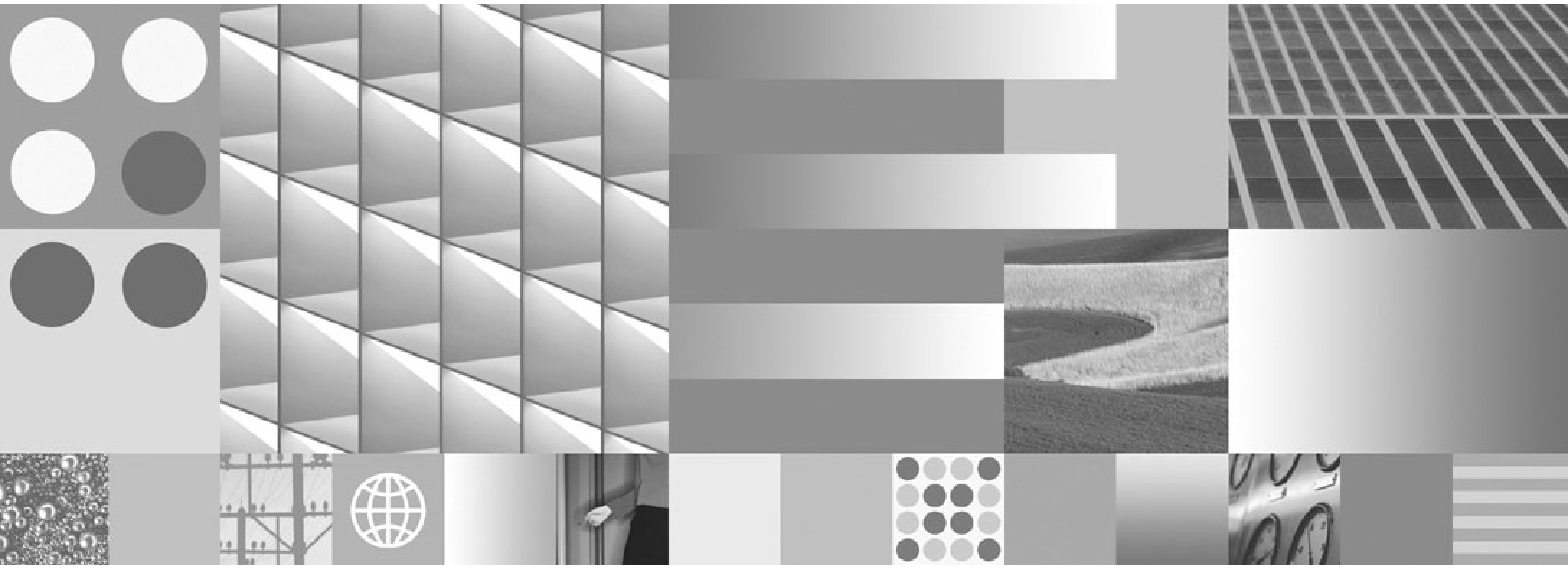
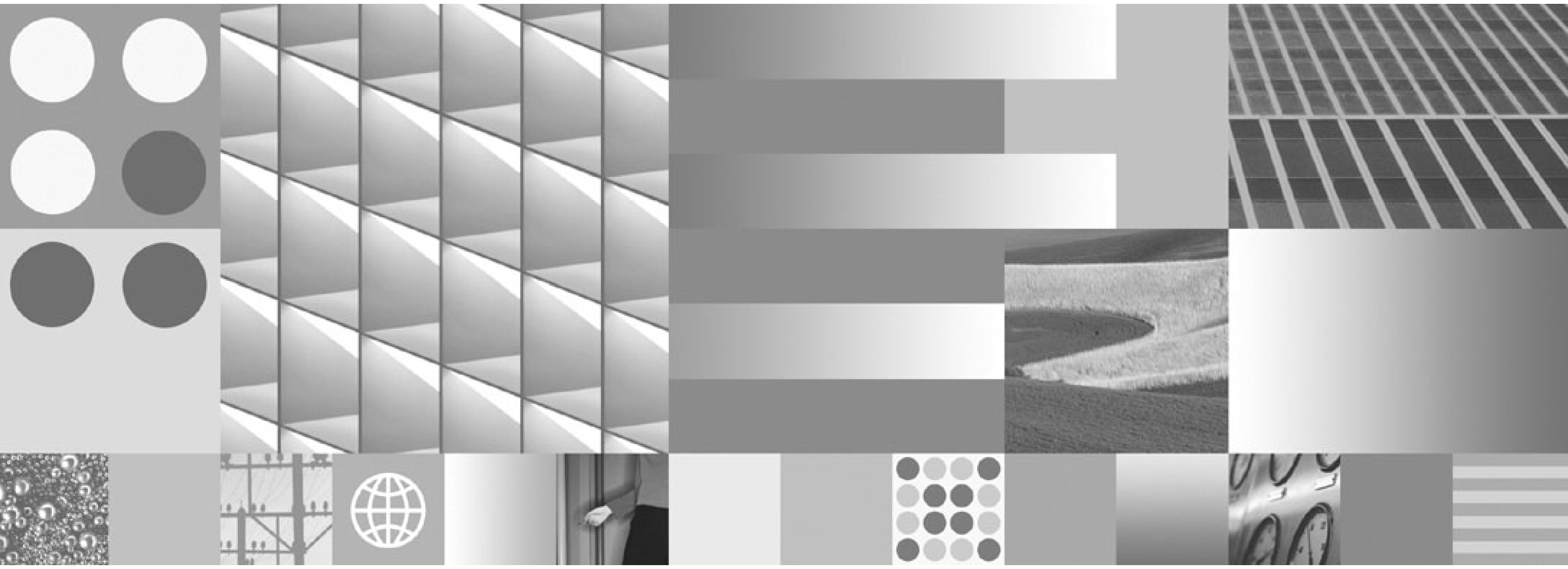
IBM DB2 9.7
for Linux, UNIX, and Windows



Visual Explain Tutorial

SC27-2462-00

IBM DB2 9.7
for Linux, UNIX, and Windows



Visual Explain Tutorial

SC27-2462-00

Note

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 95.

Edition Notice

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

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

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

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

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

Contents

Chapter 7. Visual Explain tasks 75

Part 3. Appendixes. 83

Appendix A. Overview of the DB2 technical information 85

Appendix B. Notices 95

Index 99

About this tutorial

The following tutorial provides a guide to the features of DB2® Visual Explain. By completing the lessons in this tutorial you will learn how Visual Explain lets you view the access plan for explained SQL or XQuery statements as a graph.

You will also learn to use the information available from such a graph to tune your SQL queries for better performance.

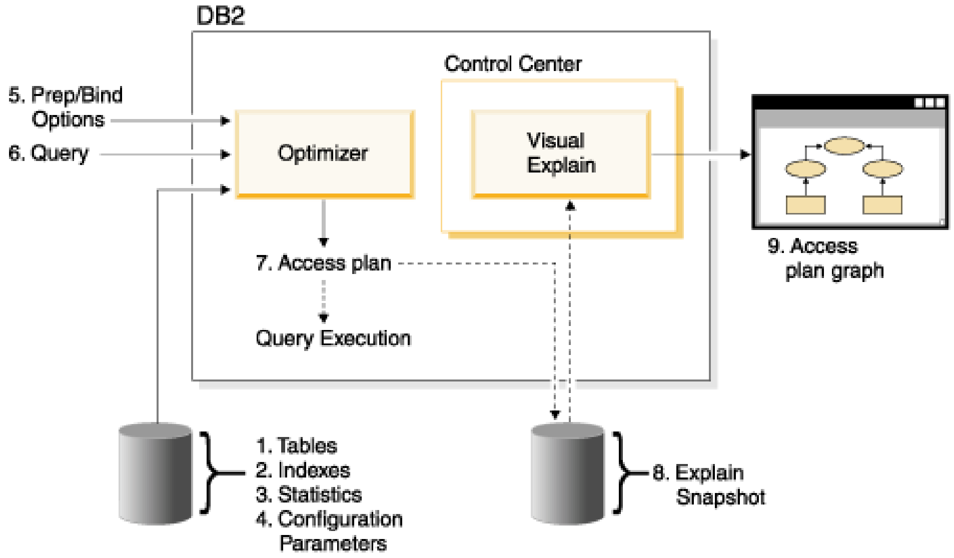
Important: Access to Visual Explain through the Control Center tools has been deprecated in Version 9.7 and might be removed in a future release. For more information, see the "Control Center tools and DB2 administration server (DAS) have been deprecated" topic in the *What's New for DB2 Version 9.7* book. Accessing Visual Explain functionality through the IBM® Data Studio toolset has not been deprecated.

Using its *optimizer*, the database manager examines your SQL queries and determines how best to access your data. This path to the data is called the *access plan*. Visual Explain enables you to see what the optimizer has done by allowing you to look at the access plan that it selected to perform a particular query. You can use Visual Explain to display the access plan as a graph. The graph is a visual presentation of the database objects involved in a query (for example, tables and indexes). It also includes the operations performed on those objects (for example, scans and sorts) and shows the flow of data.

You can improve a query's access to data by performing any or all of the following tuning activities:

1. Tune your table design and reorganizing table data.
2. Create appropriate indexes.
3. Use the **RUNSTATS** command to provide the optimizer with current statistics.
4. Choose appropriate configuration parameters.
5. Choose appropriate bind options.
6. Design queries to retrieve only required data.
7. Work with an access plan.
8. Create explain snapshots.
9. Use an access plan graph to improve an access plan.

These performance-related activities correspond to those shown in the following illustration. (Broken lines indicate actions that are required for Visual Explain.)



Learning objectives

The tutorial contains lessons on:

- Creating explain snapshots. These are requirements for displaying access plan graphs.
- Displaying and manipulating an access plan graph.
- Performing tuning activities and examining how these improve your access plan.

  Note: Performance tuning is divided into a lesson for single-partition database environments and a lesson for partitioned database environments.

You will use the supplied SAMPLE database to work through the lessons. If it is not already created, see the section on installing the SAMPLE database in the *DB2 Information Center*.

Time required

This tutorial should take approximately 60 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Skill level

Advanced

Audience

Database administrators or application developers responsible for tuning SQL queries.

Environment-specific information

Information marked with this icon pertains only to single-partition database environments.

Information marked with this icon pertains only to partitioned database environments

Part 1. Visual Explain Tutorial

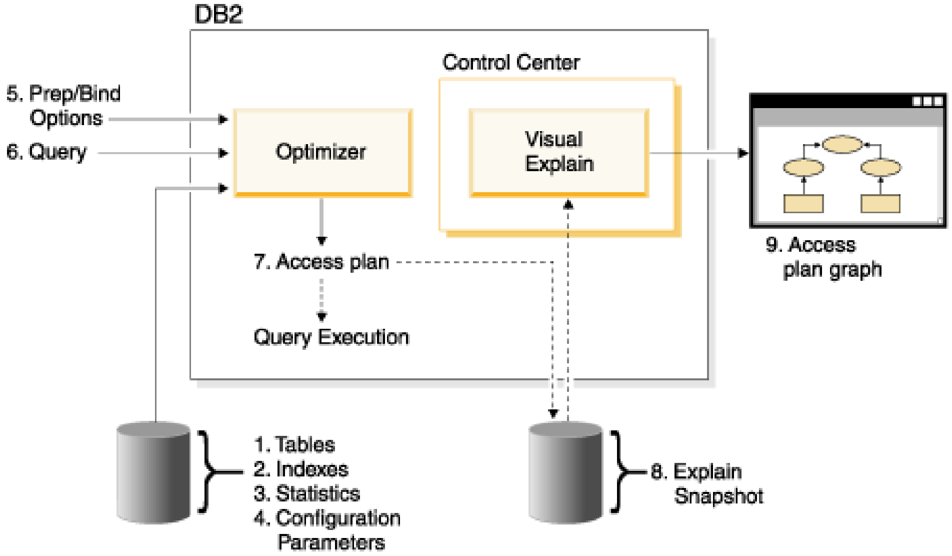
Visual Explain lets you view the *access plan* for explained SQL or XQuery statements as a graph.

Important: Access to Visual Explain through the Control Center tools has been deprecated in Version 9.7 and might be removed in a future release. For more information, see the "Control Center tools and DB2 administration server (DAS) have been deprecated" topic in the *What's New for DB2 Version 9.7* book. Accessing Visual Explain functionality through the IBM Data Studio toolset has not been deprecated.

You can use the information in the graph to tune your queries by performing the following tasks:

- Viewing the statistics that were used at the time of optimization. You can compare these statistics to the current catalog statistics to help you determine whether rebinding the package might improve performance.
- Determining whether or not an index was used to access a table. If an index was not used, Visual Explain helps you to determine which columns might benefit from being indexed.
- Viewing the effects of performing various types of tuning by comparing the before and after versions of the access plan graph for a query.
- Obtaining information about each operation in the "Access plan" on page 63 including the total estimated cost and number of rows retrieved (cardinality).

The following illustration shows the interaction between the DB2 optimizer and Visual Explain invoked from the Control Center. (Broken lines indicate actions that are required for Visual Explain.)



To learn how to use Visual Explain, you can work through the scenarios in the Visual Explain Tutorial.

Prerequisites

- To dynamically explain SQL or XQuery statements, you will need a minimum of INSERT privilege on the explain tables. If explain tables do not exist, they will be created when you explain the SQL or XQuery statements.
- To view the details of explained statements, including statistics, you will need a minimum of SELECT privilege on both the explain tables and on the system catalog tables.
- To change explained statements, you will need a minimum of UPDATE privilege on the explain tables.
- To remove explained statements, you will need a minimum of DELETE privilege on the explain tables.

To start Visual Explain:

- From the Control Center, right-click a database name and select either **Show Explained Statements History** or **Explain Query**.
- From the Command Editor, execute an explainable statement on the Interactive page or the Script page.
- From the Query Patroller, click **Show Access Plan** from either the Managed Queries Properties notebook or from the Historical Queries Properties notebook.

Troubleshooting Tips

- "Retrieving the access plan when using LONGDATACOMPAT" on page 81
- "Visual Explain support for earlier and later releases" on page 82

Chapter 1. Lesson 1. Creating explain snapshots

Creating explain snapshots that will help you understand the structure and potential execution performance of your SQL or XQuery statements.

The SQL explain facility is used to capture information about the environment in which a static or dynamic SQL or XQuery statement is compiled. The information captured allows you to understand the structure and potential execution performance of your SQL or XQuery statements. An explain snapshot is compressed information that is collected when an SQL or XQuery statement is explained. It is stored as a binary large object (BLOB) in the EXPLAIN_STATEMENT table and contains the following information:

- The internal representation of the access plan, including its operators and the tables and indexes accessed.
- The decision criteria used by the optimizer, including statistics for database objects and the cumulative cost for each operation.

In order to display an access plan graph, Visual Explain requires the information contained in an explain snapshot.

Creating the explain tables

This section demonstrates how to create explain tables.

To create explain snapshots, you must ensure that the following explain tables exist for your user ID:

- EXPLAIN_INSTANCE
- EXPLAIN_STATEMENT

To check if they exist, use the list tables command. If these tables do not exist, you must create them using the following instructions:

1. If the DB2 database manager has not already been started, issue the **db2start** command.
2. From the CLP prompt, connect to the database that you want to use.
   For this tutorial, connect to the SAMPLE database using the **connect to sample** command.
3. Create the explain tables, using the sample command file that is provided in the EXPLAIN.DDL file. This file is located in the sqllib\misc directory.
   To run the command file, go to this directory and issue the **db2 -tf EXPLAIN.DDL** command. This command file creates explain tables that are prefixed with the connected user ID. This user ID must have CREATETAB privilege on the database, or DBADM authority.

   Note: In Version 9, the Explain Statement History window displays explained records from both the SYSTOOLS schema and the schema of the current authorization ID. You must have read privilege on the SYSTOOLS explain tables in order for Visual Explain to retrieve the SYSTOOLS records and display them in the Explain Statement History window. If you do not have read access, these records will not be displayed. Also, if you have migrated from an earlier version of the DB2 database manager, you need to run db2exmig to migrate the explain tables.

Using explain snapshots

This section demonstrates how to use explain snapshots.

Four sample snapshots are provided to help you learn about Visual Explain. Information about creating your own snapshots is provided in the following sections, but you do not need to create your own snapshots to work with this tutorial:

- Creating explain snapshots for dynamic SQL or XQuery statements
- Creating explain snapshots for static SQL or XQuery statements

The query used for the sample snapshots lists the name, department, and earnings for all non-manager employees who earn more than 90% of the highest-paid manager's salary.

```
SELECT S.ID,S.NAME,O.DEPTNAME,SALARY+COMM
FROM ORG O, STAFF S
WHERE
 O.DEPTNUM = S.DEPT AND
 S.JOB <> 'Mgr' AND
 S.SALARY+S.COMM > ALL (SELECT ST.SALARY*.9
                        FROM STAFF ST
                        WHERE ST.JOB='Mgr')
```

The query has two parts:

1. The subquery (in parentheses) produces rows of data that consists of 90% of each manager's salary. Because the subquery is qualified by ALL, only the largest value from this table is retrieved.
2. The main query joins all rows in the ORG and STAFF tables where the department numbers are the same, JOB does not equal 'Mgr', and salary plus commission is greater than the value that was returned from the subquery.

The main query contains the following three predicates:

```
1. O.DEPTNUMB = S.DEPT
2. S.JOB <> 'Mgr'
3. S.SALARY+S.COMM > ALL (SELECT ST.SALARY*.9
                          FROM STAFF ST
                          WHERE ST.JOB='Mgr')
```

These predicates represent, respectively:

1. A join predicate, which joins the ORG and STAFF tables where department numbers are equal
2. A local predicate on the JOB column of the STAFF table
3. A local predicate on the SALARY and COMM columns of the STAFF table that uses the result of the subquery.

To load the sample snapshots:

1. If the database manager has not already been started, issue the **db2start** command.
2. Ensure that explain tables exist in your database.
   To do this, follow the instructions in Creating the explain tables.
3. Connect to the database that you want to use.
   For this tutorial you will connect to the SAMPLE database. To connect to the SAMPLE database, from the CLP prompt issue the **connect to sample** command. If it is not already created, see the section on installing the SAMPLE database in the *DB2 Information Center*.

4. To import the predefined snapshots, run the DB2 command file VESAMPL.DDL
   - This file is located in the sqllib\samples\ve directory.
   - This file is located in the sqllib\samples\ve\inter directory.

   To run the command file, go to this directory and issue the **db2 -tf vesampl.ddl** command.
   - This command file must be run using the same user ID that was used to create the explain tables.
   - This command file only imports the predefined snapshots. It does not create tables or data. The tuning activities described later (for example, CREATE INDEX and runstats), will be run on tables and data in the SAMPLE database.

You are now ready to display and use access plan graphs.

Creating explain snapshots for dynamic SQL or XQuery statements

This lesson outlines how to create explain snapshots for dynamic SQL or XQuery statements

Note: The creating explain snapshot information in this section is provided for your reference. Since you are provided with sample explain snapshots, it is not necessary to complete this task in order to work through the tutorial.

Follow these steps to create an *explain snapshot* for a dynamic SQL or XQuery statement:

1. If the database manager has not already been started, issue the **db2start** command.
2. Ensure that explain tables exist in your database.
   To do this, follow the instructions in “Creating the explain tables” on page 3.
3. From the CLP prompt, connect to the database that you want to use.
   For example, to connect to the SAMPLE database, issue the **connect to sample** command. To create the SAMPLE database, see the section on installing the SAMPLE database in the *DB2 Information Center*.
4. Create an explain snapshot for a dynamic SQL or XQuery statement, using either of the following commands from the CLP prompt:
   - To create an explain snapshot without executing the SQL or XQuery statement, issue the set current explain snapshot=explain command.
   - To create an explain snapshot and execute the SQL or XQuery statement, issue the set current explain snapshot=yes command.

   This command sets the explain special register. Once it is set, all subsequent SQL or XQuery statements are affected. For more information, see the CURRENT EXPLAIN SNAPSHOT special register and the SET CURRENT EXPLAIN SNAPSHOT statement.
5. Submit your SQL or XQuery statements from the CLP prompt.
6. To view the access plan graph for the snapshot, refresh the Explained Statements History window (available from the Control Center), and double-click on the snapshot.
7. Optional. To turn off the snapshot facility, issue the set current explain snapshot=no command after you submit your SQL or XQuery statements.

Creating explain snapshots for static SQL or XQuery statements

This lesson outlines how to create explain snapshots for static SQL or XQuery statements

Note: The creating explain snapshot information in this section is provided for your reference. Since you are provided with sample explain snapshots, it is not necessary to complete this task in order to work through the tutorial.

Follow these steps to create an *explain snapshot* for a static SQL or XQuery statement:

1. If the database manager has not already been started, issue the **db2start** command.
2. Ensure that explain tables exist in your database.
   To do this, follow the instructions in “Creating the explain tables” on page 3.
3. From the CLP prompt, connect to the database that you want to use.
   For example, to connect to the SAMPLE database, issue the **connect to sample** command.
4. Create an explain snapshot for a static SQL or XQuery statement by using the EXPLSNAP option when binding or preparing your application.
   For example, issue the bind *your file* explsnap yes command.
5. Optional. To view the access plan graph for the snapshot, refresh the Explained Statements History window (available from the Control Center), and double-click on the snapshot.

For information about using the EXPLSNAP option see “Bind considerations”, CURRENT EXPLAIN SNAPSHOT special register, the BIND and REBIND commands, and the EXPLAIN statement.

What’s Next

Moving on to lesson 2.

In ″Lesson 2. Displaying and using an access plan graph,″ you will learn how to view an access plan graph and understand its contents.

Chapter 2. Lesson 2. Displaying and using an access plan graph

In this lesson, you will use the Access Plan Graph window to display and use an access plan graph.

An access plan graph is a graphical representation of an access plan. From it, you can view the details for:

- Tables (and their associated columns) and indexes
- Operators (such as table scans, sorts, and joins)
- Table spaces and functions

You can display an access plan graph by:

- Choosing from a list of previously explained statements.
- Choosing from a list of explainable statements in a package.
- Dynamically explaining as SQL or XQuery statement.

Because you will be working with the access plan graphs for the sample explain snapshots that you loaded in Lesson 1, you will choose from a list of previously explained statements. For information on the other methods of displaying access plan graphs refer to the Visual Explain Help.

Displaying an access plan graph by choosing from a list of previously explained SQL or XQuery statements

The graph is read from bottom to top. The first step of the query is listed at the bottom of the graph and the last step is listed at the top.

To display an access plan graph by choosing from a list of previously explained statements:

1. In the Control Center, expand the object tree until you find the SAMPLE database.
2. Right-click on the database and select **Show explained statements history** from the pop-up menu.
   The Explained Statements History window opens.
3. You can only display an access plan graph for a statement that has an explain snapshot. Statements that qualify will have an entry of YES in the **Explain Snapshot** column. Double-click on the entry identified as Query Number 1 (you might need to scroll to the right to find the **Query Number** column).
   The Access Plan Graph window for the statement opens.

Reading the symbols in an access plan graph

The access plan graph shows the structure of an access plan as a tree.

The *nodes* of the tree represent:

- Tables, shown as rectangles
- Indexes, shown as diamonds
- Operators, shown as octagons. TQ operators, shown as parallelograms

- Table functions, shown as hexagons

For operators, the number in brackets to the right of the operator type, is a unique identifier for each node. The number below the operator type, is the cumulative *cost*.

Using the zoom slider to magnify parts of a graph

This section outlines the use of the zoom slider to magnify parts of the graph.

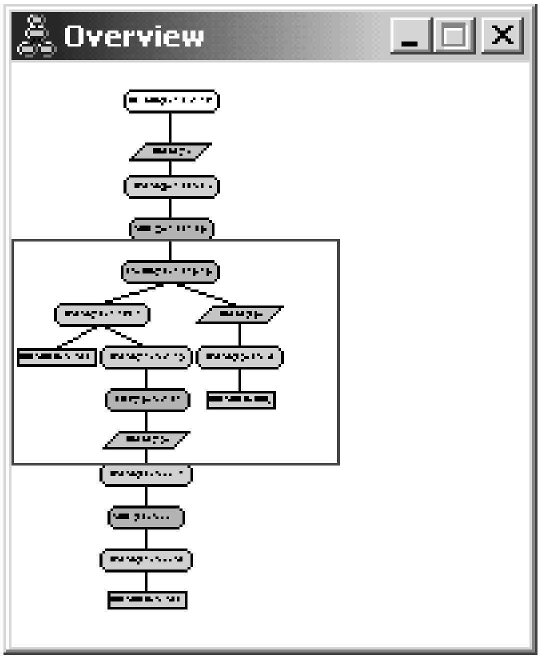
When you display an access plan graph, the entire graph is shown, and you may not be able to see the details that distinguish each *node*.

From the Access Plan Graph window, use the **zoom slider** to magnify parts of a graph:

1. Position the mouse pointer over the small scroll box in the Zoom slider bar at the left side of the graph.
2. Left-click and drag the slider until the graph is at the level of magnification you want.

   To view different parts of the graph, use the scroll bar.

   To view a large and complicated access plan graph, use the Graph Overview window. You can use this window to see which part of the graph you are viewing, and to zoom in on or scroll through the graph. The section in the zoom box is shown in the access plan.

   To scroll through the graph, position the mouse pointer over the highlighted area in the Graph Overview window, press and hold mouse button 1, then move the mouse until you see the part of the access graph you want.

Getting more details about the objects in a graph

This section shows you how to access more information about the objects in an access plan graph.

You can access more information about the objects in an access plan graph. You can display:

- System catalog statistics for objects such as:
  - Tables, indexes, or table functions
  - Information about operators, such as their costs, properties, and input arguments
  - Built-in functions or user-defined functions
  - Table spaces
  - Columns referenced in an SQL or XQuery statement
- Information about configuration parameters and bind options (optimizing parameters).

Getting statistics for tables, indexes, and table functions

To view catalog statistics for a single table (rectangular), index (diamond), or table function (hexagon) in a graph, double-click on its *node*. A Statistics window opens for the selected objects, displaying information about the statistics that were in effect at the time the snapshot was created, as well as those that currently exist in the system catalog tables.

To view catalog statistics for *multiple* tables, indexes, or table functions in a graph, select each one by clicking on it (it is high-lighted); then select **Node->Show Statistics**. A statistics window opens for each of the selected objects. (The window might be stacked and some dragging and dropping might be required in order to access them all.)

If the entry for *STATS_TIME* in the **Explained** column contains the entry *Statistics not updated*, then no statistics existed when the optimizer created the access plan. Therefore, if the optimizer required certain statistics to create an access plan, it used defaults. If default statistics were used by the optimizer, they are identified as (default) in the **Explained** column.

Getting details about operators in a graph

To view catalog statistics for a single operator (octagon), double-click on its *node*. An Operator details window opens for the selected operator, displaying information such as:

- The estimated cumulative costs (I/O, CPU instructions, and total cost)
- The cardinality (that is, the estimated number of rows searched) so far
- Tables that have been accessed and joined so far in the plan
- Columns of those tables that have been accessed so far
- Predicates that have been applied so far, including their estimated *selectivity*
- The input arguments for each operator

To view details for *multiple* operators, select each one by clicking on it (it is highlighted); then select **Node->Show Details**. A Statistics window opens for each of the selected objects. (The windows might be stacked and some dragging and dropping might be required in order to access them all.)

Getting statistics for functions

To view catalog statistics for built-in functions and user-defined functions, select **Statement->Show statistics->Functions**. Select one or more entries from the list displayed on the Functions window. A Function Statistics window opens for each of the selected functions.

Getting statistics for table spaces

To view catalog statistics for table spaces, select **Statement->Show statistics->Table spaces**. Select one or more entries from the list displayed on the Table Spaces window. A Table Space Statistics window opens for each of the selected table spaces.

Getting statistics for columns in an SQL or XQuery statement

To get statistics for the columns referenced in an SQL or XQuery statement:

1. Double-click on a table in the access plan graph.The Table Statistics window opens.
2. Click the **Referenced Columns** push button. The Referenced Columns window opens, listing that columns in the table.
3. Select one or more columns from the list.A Referenced Column Statistics window opens for each of the columns selected.

Getting information about configuration parameters and bind options

To view information about configuration parameters and bind options (optimization parameters), select **Statement->Show optimization parameters** from the Access Plan Graph window. The Optimization Parameters window opens, displaying information about the parameter values that were in effect at the time the snapshot was created, as well as the current values.

Changing the appearance of a graph

This section outlines the steps required to change the appearance of a graph.

To change various characteristics of how a graph appears:

1. From the Access Plan Graph window, select **View->Settings**. The Access Plan Graph Settings notebook opens.
2. To change the background color, choose the Graph tab.
3. To change the color of various operators, use the Basic, Extend, Update, and Miscellaneous tabs.
4. To change to color of table, index, or table function nodes, select the Operand tab.
5. To specify which information is shown in operator nodes (types of *cost* or *cardinality*, which is the estimated number of rows returned so far), choose the Operator tab.
6. To specify whether the schema names or user IDs are shown in the table nodes, select the Operand tab.
7. To specify whether nodes are shown two-dimensionally or three-dimensionally, select the Node tab.

8. To update the graph with the options you chose and save the settings, click **Apply**.

What's Next

Moving on to lesson 3 or 4.

If you are working in a single-partition database environment, go to Chapter 3, "Lesson 3. Improving an access plan in a single-partition database environment," on page 13, where you will learn how different tuning activities can change and improve an access plan.

If you are working in a partitioned database environment, go to Chapter 4, "Lesson 4. Improving an access plan in a partitioned database environment," on page 31, where you will learn how different tuning activities can change and improve an access plan.

Chapter 3. Lesson 3. Improving an access plan in a single-partition database environment

In this lesson, you will learn how the access plan and related windows for the basic query change when you perform various tuning activities.

Using a series of examples, accompanied by illustrations, you will learn how the estimated total cost for the access plan of even a simple query can be improved by using the **RUNSTATS** command and adding appropriate indexes.

As you gain experience with Visual Explain, you will discover other ways to tune queries.

Working with access plan graphs

Using the four sample explain snapshots as examples, you will learn how tuning is an important part of database performance.

The queries associated with the explain snapshots are numbered 1 - 4. Each query uses the same SQL or XQuery statement (described in Lesson 1):

```
SELECT S.ID,S.NAME,O.DEPTNAME,SALARY+COMM
FROM ORG O, STAFF S
WHERE
 O.DEPTNUMB = S.DEPT AND
 S.JOB <> 'Mgr' AND
 S.SALARY+S.COMM > ALL ( SELECT ST.SALARY*.9
                         FROM STAFF ST
                         WHERE ST.JOB='Mgr' )
```

But each iteration of the query uses more tuning technics than the previous execution.

For example, Query 1 has had no performance tuning, while Query 4 has had the most. The difference in the queries are described below:

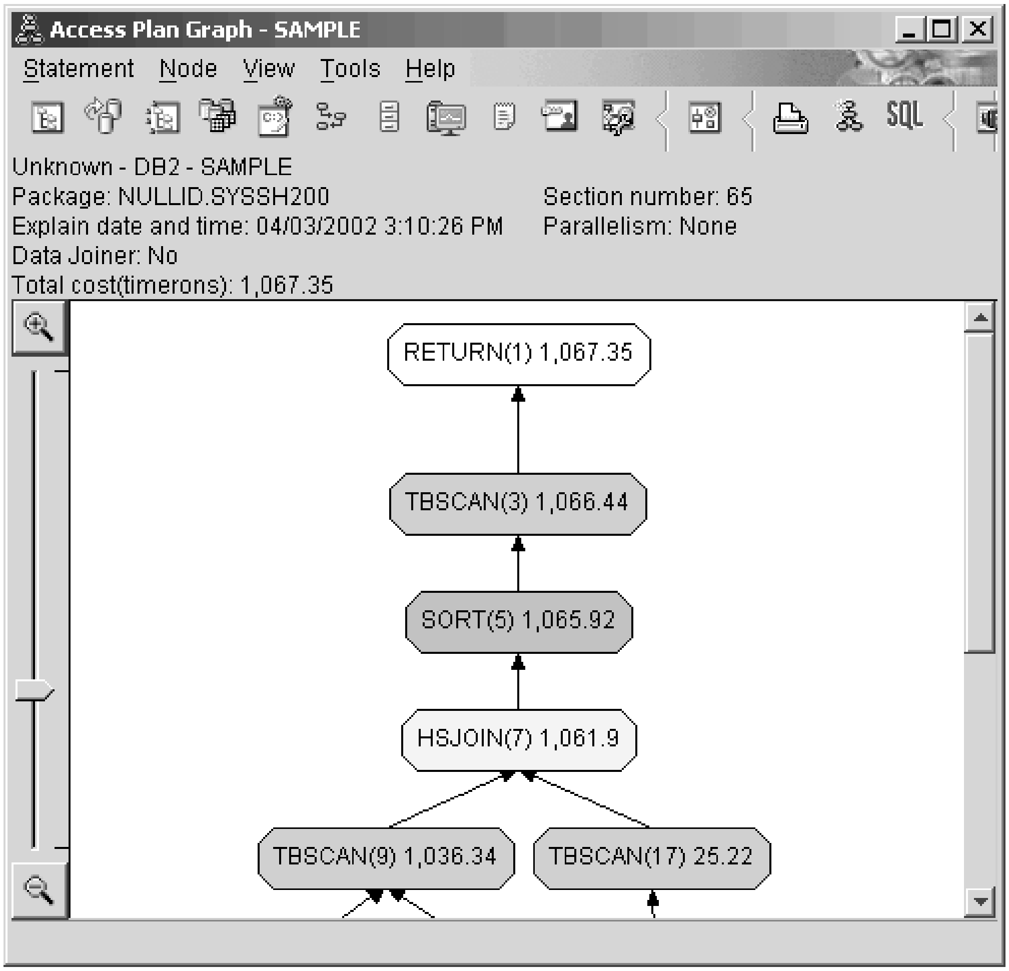
Query 1
Running a query with no indexes and no statistics
Query 2
Collecting current statistics for the tables and indexes in a query
Query 3
Creating indexes on columns used to join tables in a query
Query 4
Creating additional indexes on table columns

Running a query with no indexes and no statistics in a single-partition database environment

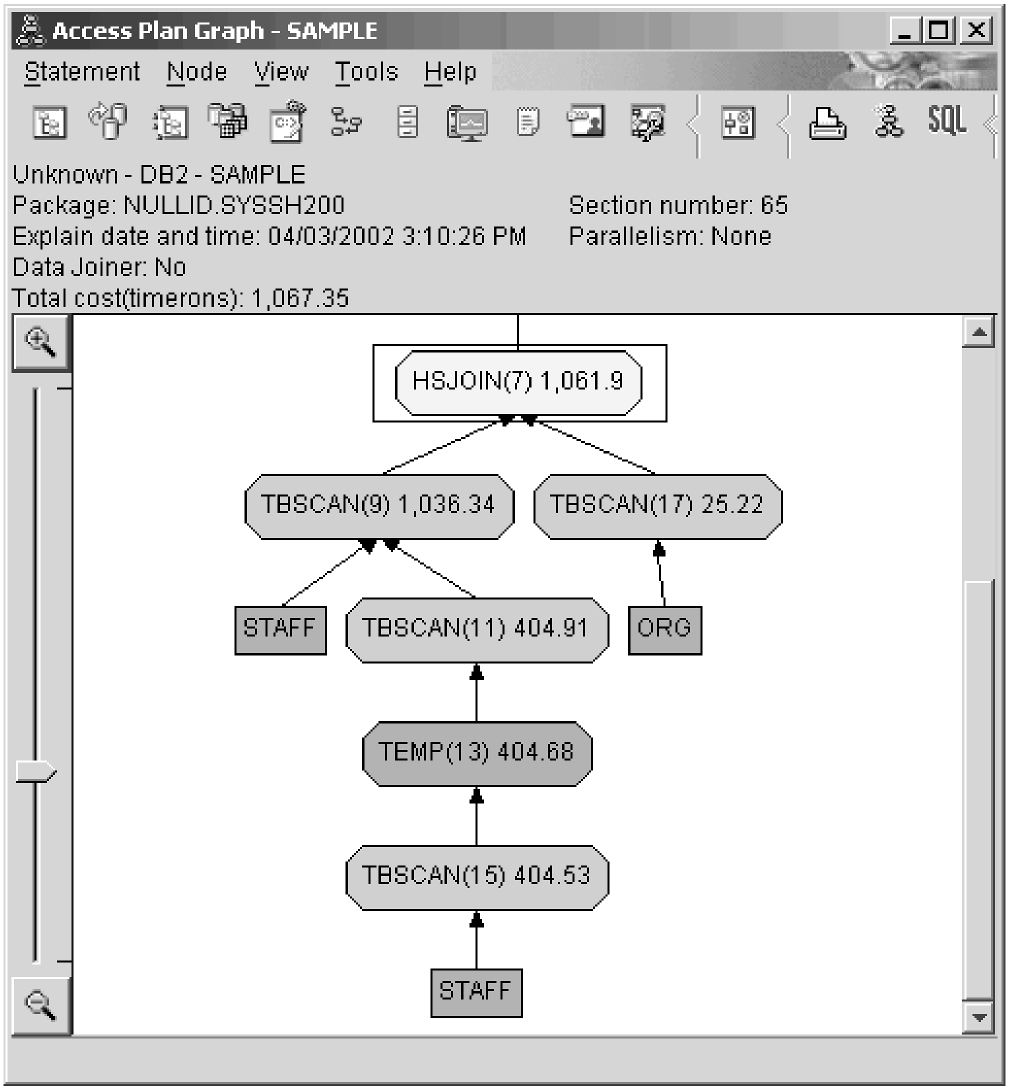
In this example, the access plan was created for the SQL query with no indexes and no statistics.

To view the access plan graph for this query (Query 1):

1. In the Control Center, expand the object tree until you find the SAMPLE database.
2. Right-click on the database and select **Show explained statements history** from the pop-up menu.
   The Explained Statements History window opens.
3. Double-click on the entry identified as Query Number 1 (you might need to scroll to the right to find the **Query Number** column)
   The Access Plan Graph window for the statement opens.



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 3:10:26 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 1,067.35



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Explain date and time: 04/03/2002 3:10:26 PM
Data Joiner: No
Total cost(timerons): 1,067.35
Section number: 65
Parallelism: None

Answering the following questions will help you understand how to improve the query.

1. Do current statistics exist for each table in the query?

   To check if current statistics exist for each table in the query, double-click each table node in the access plan graph. In the Table Statistics window that opens, the **STATS_TIME** row under the **Explained** column contains the words ″Statistics not updated″ if no statistics had been collected at the time when the snapshot was created.

   If current statistics do not exist, the optimizer uses default statistics, which may differ from the actual statistics. Default statistics are identified by the word ″default″ under the **Explained** column in the Table Statistics window.

   According to the information in the Table Statistics window for the ORG table, the optimizer used default statistics (as indicated next to the explained values). Default statistics were used because actual statistics were not available when the snapshots were created (as indicated in the **STATS_TIME** row).

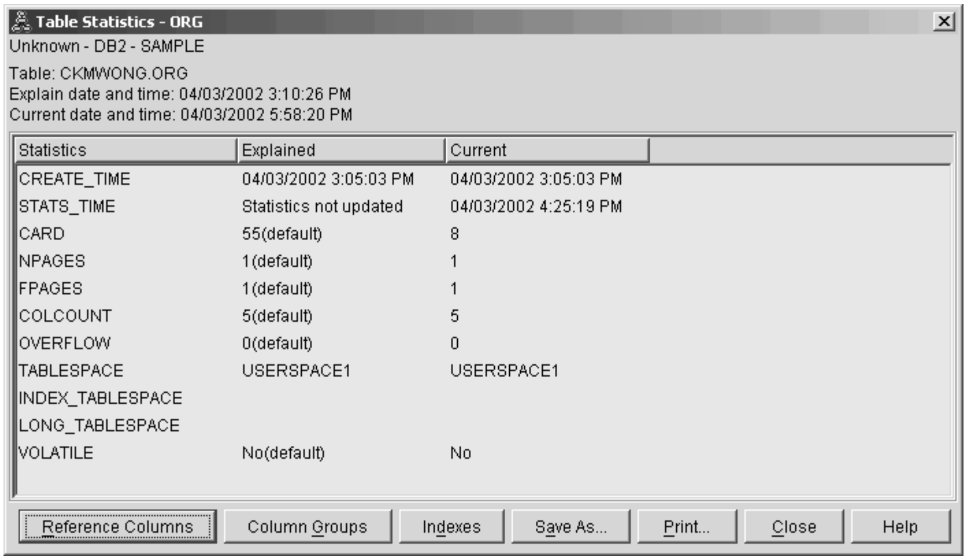


Table Statistics - ORG

Unknown - DB2 - SAMPLE

Table: CKMWONG.ORG
Explain date and time: 04/03/2002 3:10:26 PM
Current date and time: 04/03/2002 5:58:20 PM

Statistics	Explained	Current
CREATE_TIME	04/03/2002 3:05:03 PM	04/03/2002 3:05:03 PM
STATS_TIME	Statistics not updated	04/03/2002 4:25:19 PM
CARD	55(default)	8
NPAGES	1(default)	1
FPAGES	1(default)	1
COLCOUNT	5(default)	5
OVERFLOW	0(default)	0
TABLESPACE	USERSPACE1	USERSPACE1
INDEX_TABLESPACE		
LONG_TABLESPACE		
VOLATILE	No(default)	No

2. Does this access plan use the most effective methods of accessing data?

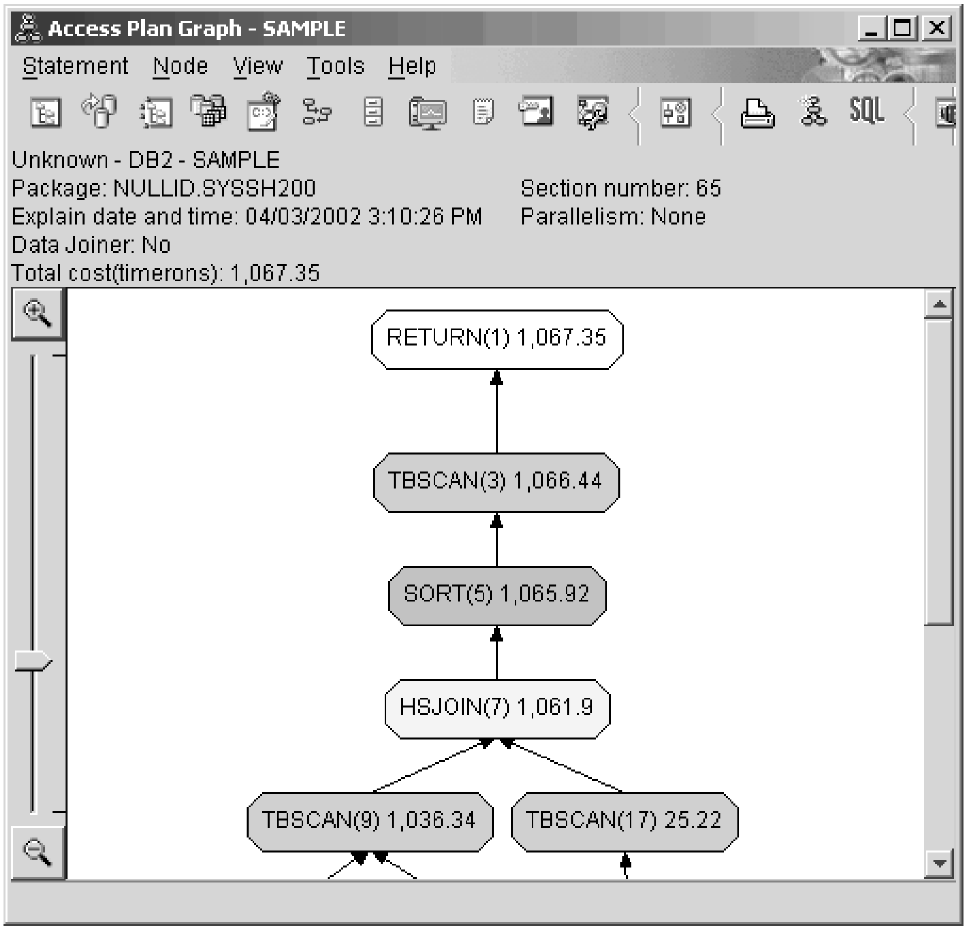
   This access plan contains table scans, not index scans. Table scans are shown as octagons and are labeled "TBSCAN operator" on page 56. If Index scans had been used they would appear as diamonds and would be labeled "IXSCAN" on page 52. The use of an index that was created for a table is more cost-effective than a table scan if small amounts of data are being extracted.

3. How effective is this access plan?

   You can determine the effectiveness of an access plan only if it is based on actual statistics. Since the optimizer used default statistics in the access plan, you cannot determine how effective the plan is.

   In general, you should make a note of the total estimated "Cost" on page 65 for the access plan for later comparison with revised access plans. The cost listed in each node is cumulative, from the first steps of your query up to and including the node.

   In the Access Plan Graph window, the total cost is approximately 1,067 timerons, shown in **RETURN (1)** at the top of the graph. The total estimated cost is also shown in the top area of the window.



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 3:10:26 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 1,067.35

What's Next

Moving on to query 2.

Query 2 looks at an access plan for the basic query after runstats has been run. Using the runstats command provides the optimizer with current statistics on all tables accessed by the query.

Collecting current statistics for the tables and indexes using runstats in a single-partition database environment

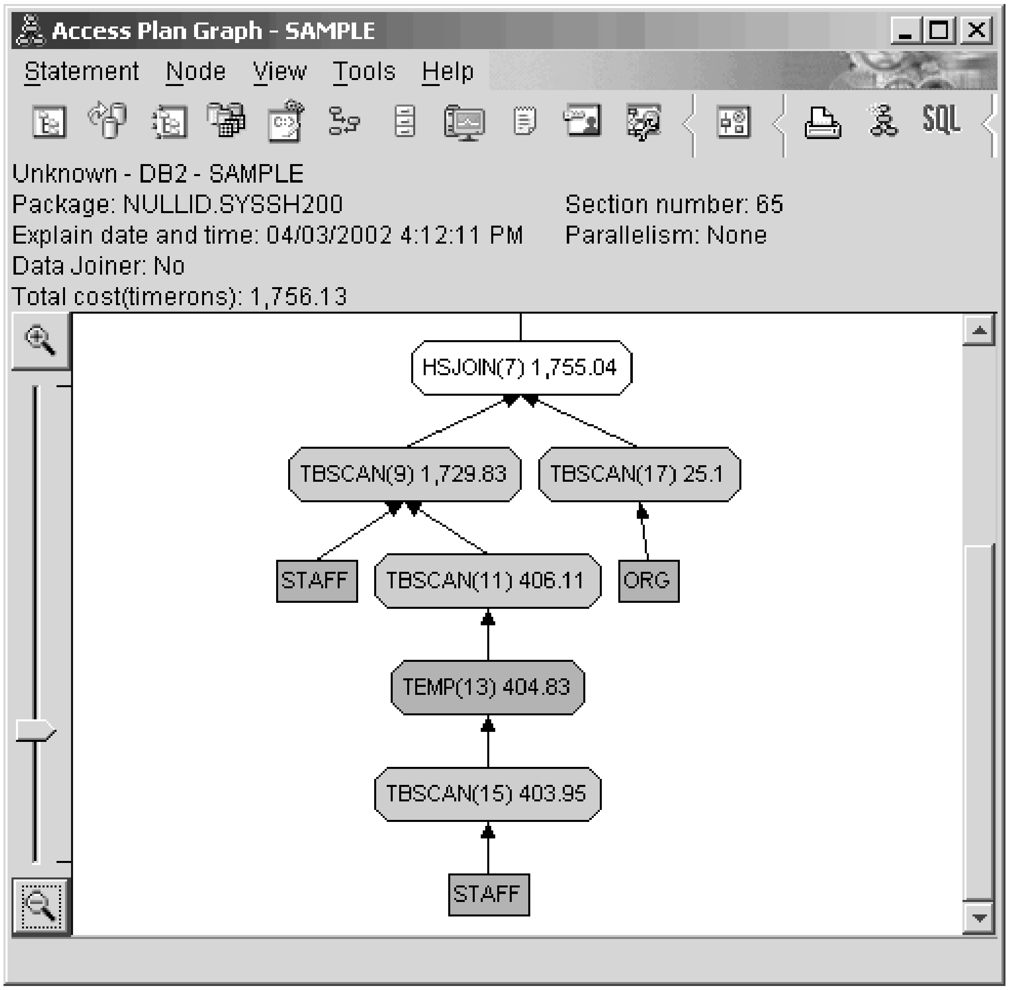
This example builds on the access plan described in Query 1 by collecting current statistics with the runstats command.

It is highly recommended that you use the runstats command to collect current statistics on tables and indexes, especially if significant update activity has occurred or new indexes have been created since the last time the runstats command was executed. This provides the optimizer with the most accurate information with which to determine the best access plan. If current statistics are not available, the optimizer can choose an inefficient access plan based on inaccurate default statistics.

Be sure to use runstats *after* making your table updates; otherwise, the table might appear to the optimizer to be empty. This problem is evident if cardinality on the

Operator Details window equals zero. In this case, complete your table updates, rerun the runstats command, and recreate the explain snapshots for affected tables.

To view the access plan graph for this query (Query 2), in the Explained Statements History window, double-click on the entry identified as Query Number 2. The Access Plan Graph window for this execution of the statement opens.



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 4:12:11 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 1,756.13

Answering the following questions will help you understand how to improve the query.

1. Do current statistics exist for each table in the query?

   The Table Statistics window for the ORG table shows that the optimizer used actual statistics (the **STATS_TIME** value is the actual time that the statistics were collected). The accuracy of the statistics depends on whether there was significant changes to the contents of the tables since the runstats command was run.

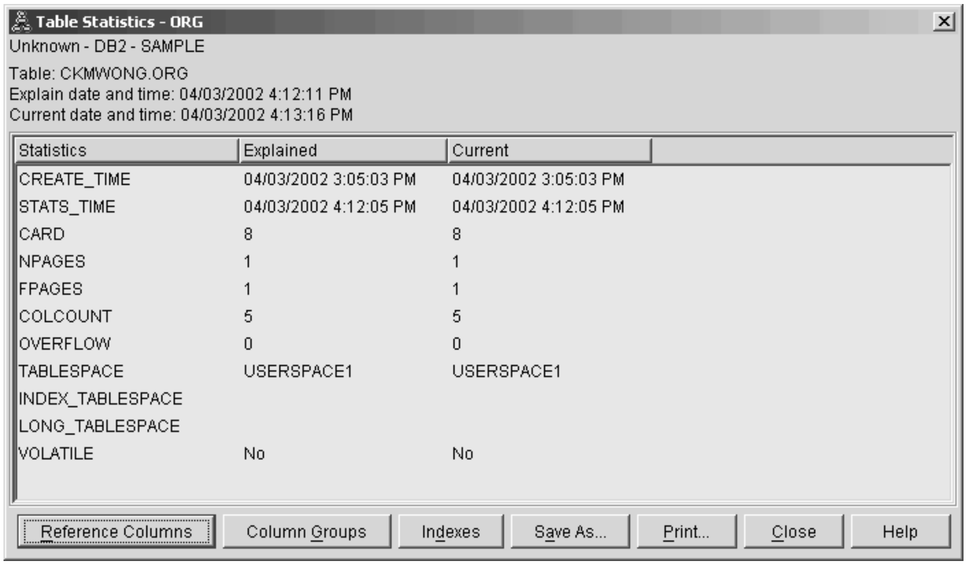


Table Statistics - ORG

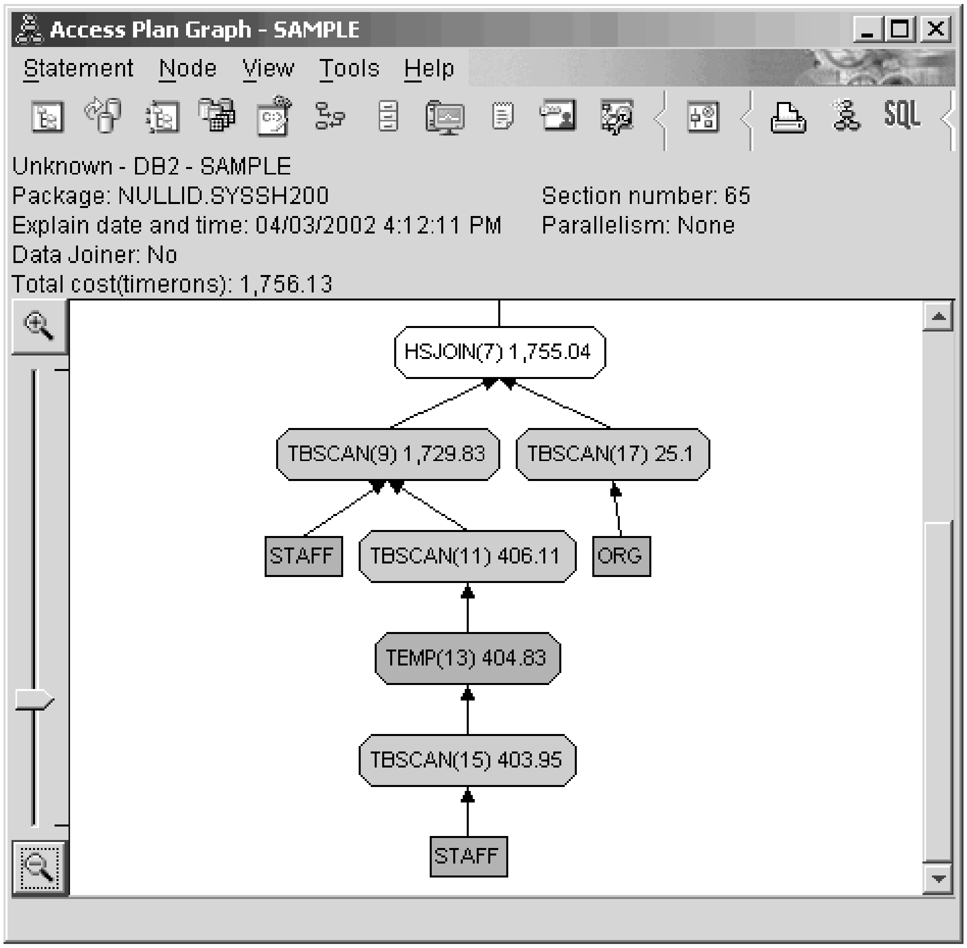
Unknown - DB2 - SAMPLE

Table: CKMWONG.ORG
Explain date and time: 04/03/2002 4:12:11 PM
Current date and time: 04/03/2002 4:13:16 PM

Statistics	Explained	Current
CREATE_TIME	04/03/2002 3:05:03 PM	04/03/2002 3:05:03 PM
STATS_TIME	04/03/2002 4:12:05 PM	04/03/2002 4:12:05 PM
CARD	8	8
NPAGES	1	1
FPAGES	1	1
COLCOUNT	5	5
OVERFLOW	0	0
TABLESPACE	USERSPACE1	USERSPACE1
INDEX_TABLESPACE		
LONG_TABLESPACE		
VOLATILE	No	No

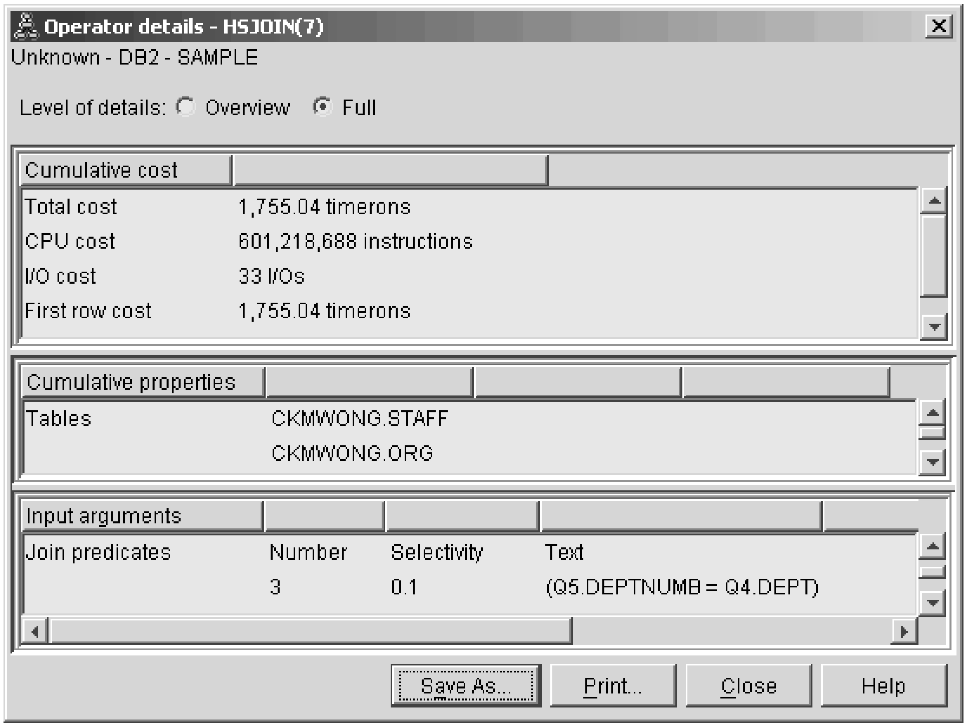
2. Does this access plan use the most effective methods of accessing data?

   Like Query 1, the access plan in Query 2 uses table scans (“TBSCAN operator” on page 56) not index scans (“IXSCAN” on page 52). Even though current statistics exist, an index scan was not done because there are no indexes on the columns that were used by the query. One way to improve the query would be to provide the optimizer with indexes on columns that are used to join tables (that is, on columns that are used in join “Predicate” on page 69). In this example, this is the first merge scan join: HSJOIN (7).



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 4:12:11 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 1,756.13

In the Operator Details window for the **HSJOIN (7)** operator, look at the **Join predicates** section under **Input arguments**. The columns used in this join operation are listed under the **Text** column. In this example, these columns are DEPTNUMB and DEPT.



Operator details - HSJOIN(7)

Unknown - DB2 - SAMPLE

Level of details: ○ Overview ● Full

Cumulative cost	
Total cost	1,755.04 timerons
CPU cost	601,218,688 instructions
I/O cost	33 I/Os
First row cost	1,755.04 timerons

Cumulative properties			
Tables	CKMWONG.STAFF		
	CKMWONG.ORG		

Input arguments			
Join predicates	Number	Selectivity	Text
	3	0.1	(Q5.DEPTNUMB = Q4.DEPT)

3. How effective is this access plan?

   Access plans based on up-to-date statistics always produce a realistic estimated cost (measured in timerons). Because the estimated cost in Query 1 was based on default statistics, the cost of the two access plan graphs cannot be compared to determine which one is more effective. Whether the cost is higher or lower is not relevant. You must compare the cost of access plans that are based on actual statistics to get a valid measurement of effectiveness.

What’s Next

Moving on to query 3.

Query 3 looks at the effects of adding indexes on the DEPTNUMB and DEPT columns. Adding indexes on the columns that are used in join predicates can improve performance.

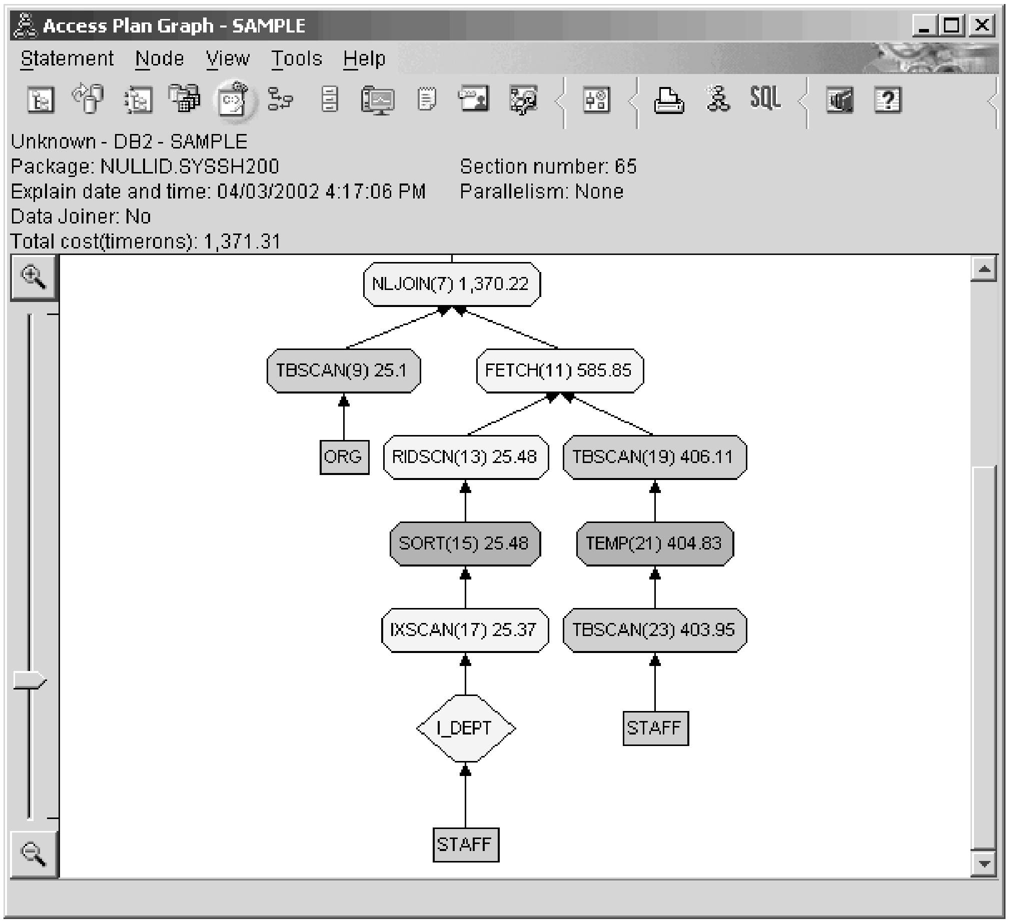
Creating indexes on columns used to join tables in a query in a single-partition database environment

This example builds on the access plan described in Query 2 by creating indexes on the DEPT column on that STAFF table and on the DEPTNUMB column on the ORG table.

Note: Recommended indexes can be created using the Design Advisor.

To view the access plan graph for this query (Query 3): in the Explained Statements History window, double-click the entry identified as Query Number 3. The Access Plan Graph window for this execution of the statement opens.

Note: Even though an index was created for DEPTNUM, the optimizer did not use it.



Unknown - DB2 - SAMPLE

Package: NULLID.SYSSH200

Section number: 65

Explain date and time: 04/03/2002 4:17:06 PM

Parallelism: None

Data Joiner: No

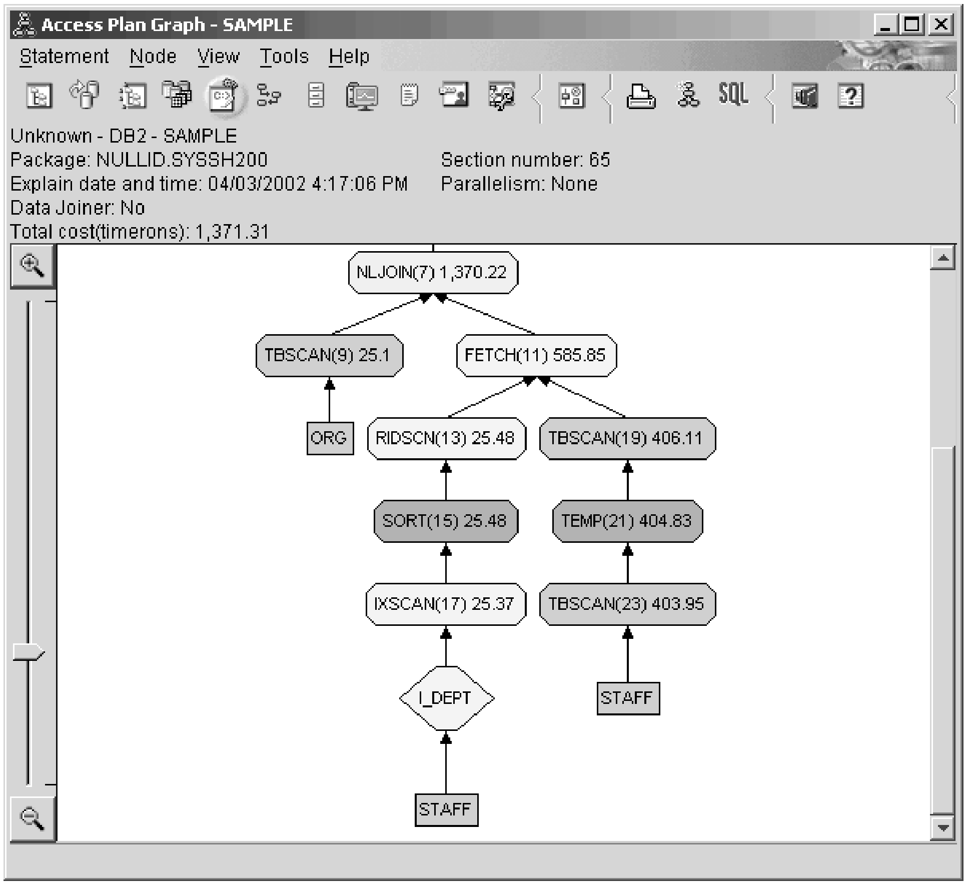
Total cost(timerons): 1,371.31

Answering the following questions will help you understand how to improve the query.

1. What has changed in the access plan with indexes?

   A “NLJOIN operator” on page 53, NLJOIN (7), has replaced the merge scan join HSJOIN (7) that was used in Query 2. Using a nested loop join resulted in a lower estimated cost than a merge scan join because this type of join does not require any sort or temporary tables.

   A new diamond-shaped node, **I_DEPT**, has been added just above the STAFF table. This node represents the index that was created on DEPT, and it shows that the optimizer used an index scan instead of a table scan to determine which rows to retrieve.



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 4:17:06 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 1,371.31

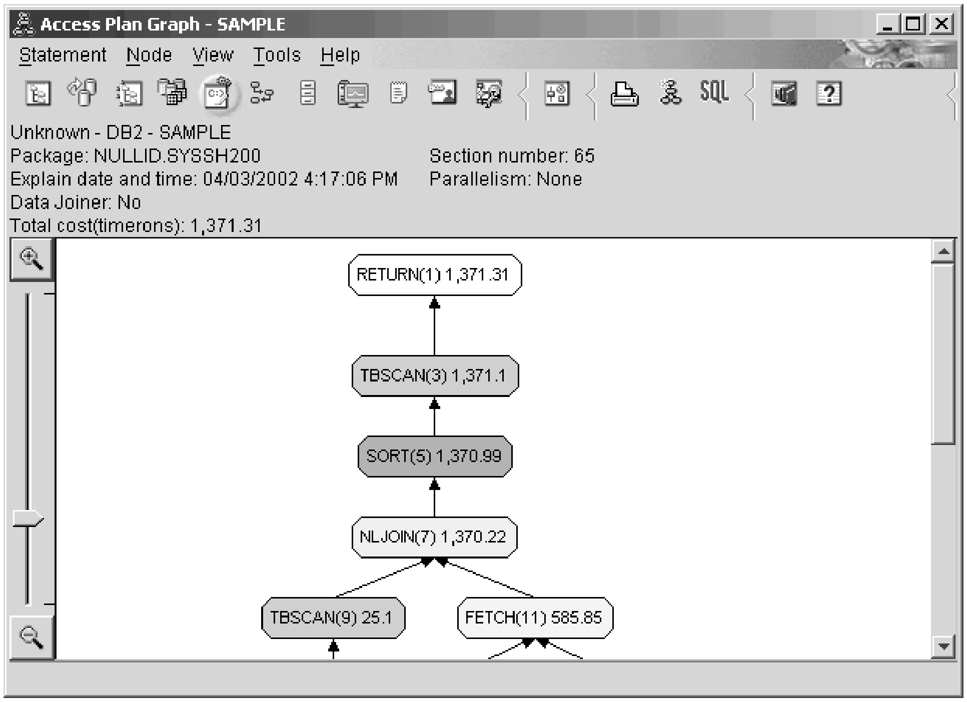
In this portion of the access plan graph, notice that a new index (I_DEPT) was created on the DEPT column and IXSCAN (17) was used to access the STAFF table. In Query 2, a table scan was used to access the STAFF table.

2. Does this access plan use the most effective methods of accessing data?

   As a result of adding indexes, an “IXSCAN” on page 52 node, IXSCAN (17), was used to access the STAFF table. Query 2 did not have an index; therefore, a table scan was used in that example.

   The “FETCH” on page 50 node, FETCH (11), shows that in addition to using the index scan to retrieve the column DEPT, the optimizer retrieved additional columns from the STAFF table, using the index as a pointer. In this case, the combination of index scan and fetch is calculated to be less costly than the full table scan used in the previous access plans.

   Note: The node for the STAFF table appears twice, to show its relationship both to the index for DEPT and to the FETCH operation.

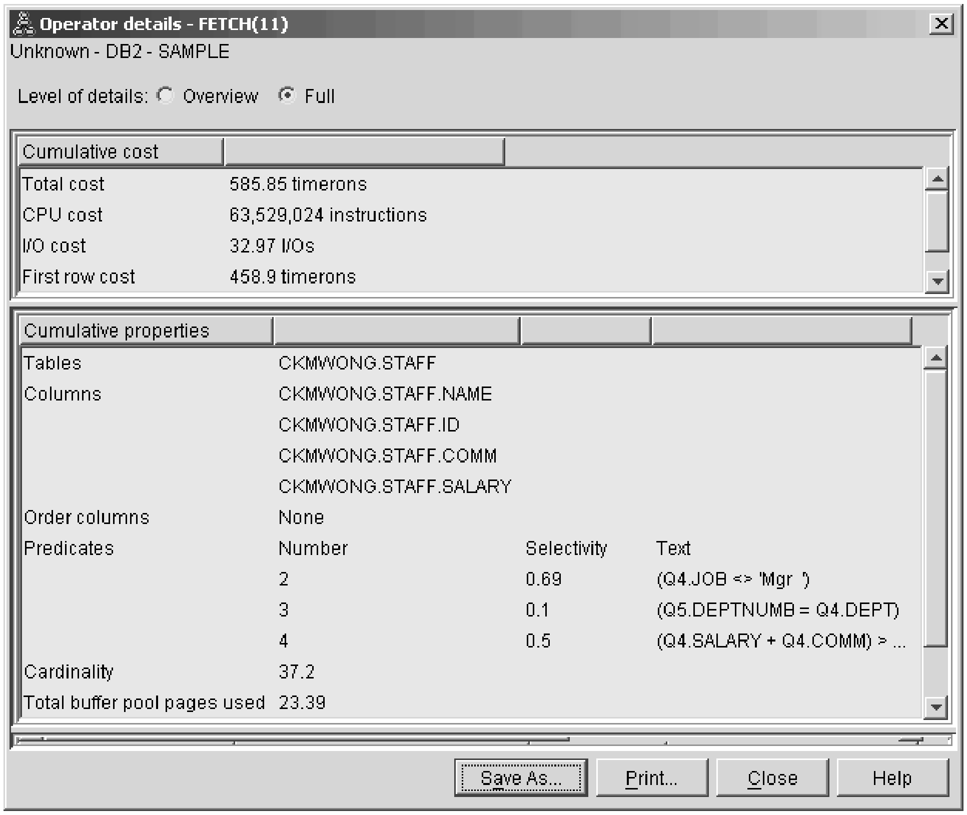


Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 4:17:06 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 1,371.31

The access plan for this query shows the effect of creating indexes on columns involved in join predicates. Indexes can also speed up the application of local predicates. Let’s look at the local predicates for each table in this query to see how adding indexes to columns referenced in local predicates might affect the access plan.

In the Operator Details window for the FETCH (11) operator, look at the columns under **Cumulative Properties**. The column used in the predicate for this fetch operation is JOB, as shown in the Predicates section.

Note: The selectivity of this predicate is .69. This means that with this predicate, 69% of the rows will be selected for further processing.



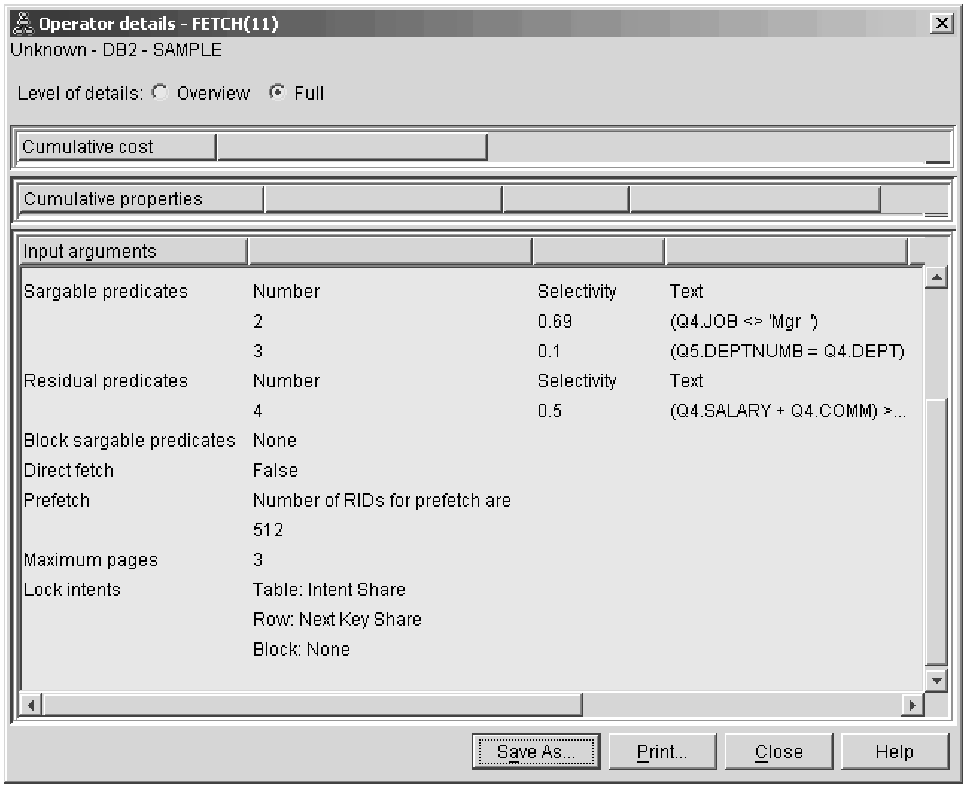
Operator details - FETCH(11)

Unknown - DB2 - SAMPLE

Level of details: ○ Overview ● Full

Cumulative cost	
Total cost	585.85 timerons
CPU cost	63,529,024 instructions
I/O cost	32.97 I/Os
First row cost	458.9 timerons

Cumulative properties			
Tables	CKMWONG.STAFF		
Columns	CKMWONG.STAFF.NAME		
	CKMWONG.STAFF.ID		
	CKMWONG.STAFF.COMM		
	CKMWONG.STAFF.SALARY		
Order columns	None		
Predicates	Number	Selectivity	Text
	2	0.69	(Q4.JOB <> 'Mgr ')
	3	0.1	(Q5.DEPTNUMB = Q4.DEPT)
	4	0.5	(Q4.SALARY + Q4.COMM) > ...
Cardinality	37.2		
Total buffer pool pages used	23.39		



Operator details - FETCH(11)

Unknown - DB2 - SAMPLE

Level of details: ○ Overview ● Full

Cumulative cost

Cumulative properties

Input arguments			
Sargable predicates	Number	Selectivity	Text
	2	0.69	(Q4.JOB <> 'Mgr ')
	3	0.1	(Q5.DEPTNUMB = Q4.DEPT)
Residual predicates	Number	Selectivity	Text
	4	0.5	(Q4.SALARY + Q4.COMM) >...
Block sargable predicates	None		
Direct fetch	False		
Prefetch	Number of RIDs for prefetch are		
	512		
Maximum pages	3		
Lock intents	Table: Intent Share		
	Row: Next Key Share		
	Block: None		

The Operator Details window for the FETCH (11) operator shows the columns being used in this operation. You can see that DEPTNAME is listed in the first row beside **Columns retrieve** under **Input arguments**.

3. How effective is this access plan?

This access plan is more cost effective than the one from the previous example. The cumulative cost has been reduced from approximately 1,755 timerons in Query 2 to approximately 959 timerons in Query 3.

However, the access plan for Query 3 shows an index scan IXSCAN (17) and a FETCH (11) for the STAFF table. While an index scan combined with a fetch operation is less costly than a full table scan, it means that for each row retrieved, the table is accessed once and the index is accessed once. Let's try to reduce this double access on the STAFF table.

What's Next

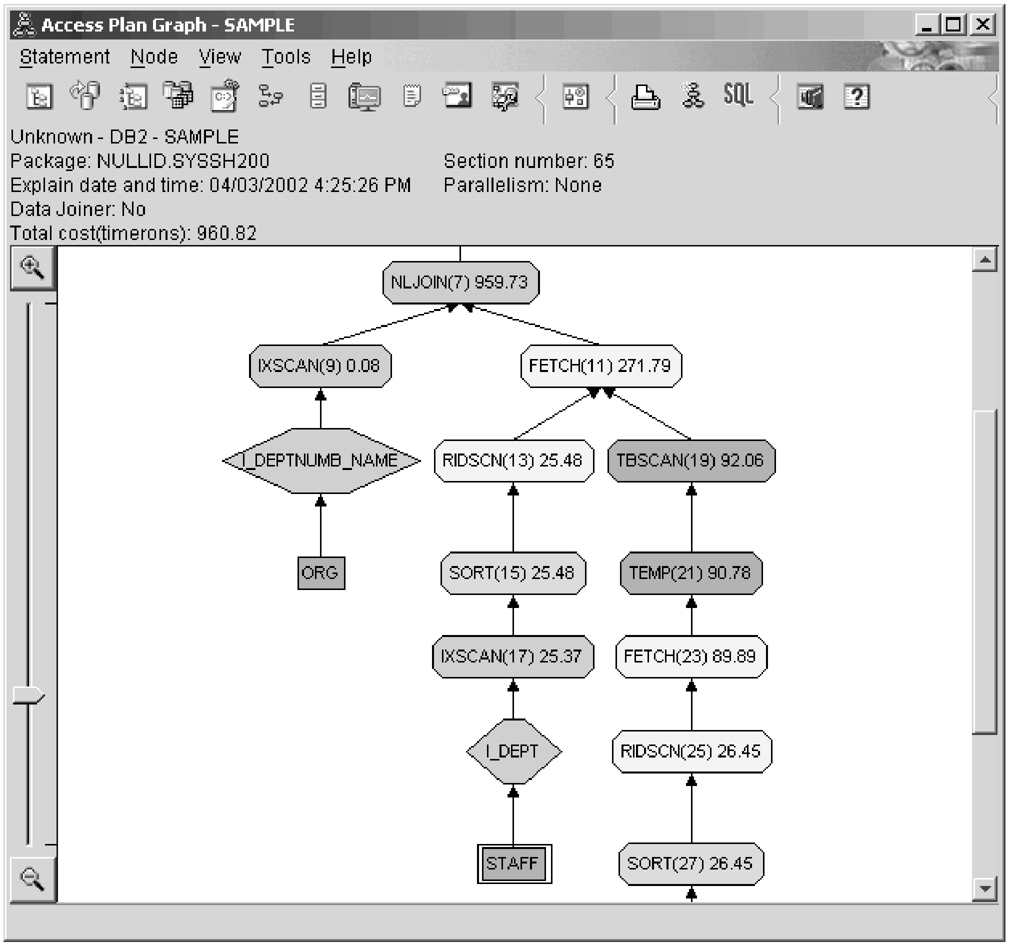
Moving on to query 4.

Query 4 reduces the fetch and index scan to a single index scan without a fetch. Creating additional indexes might reduce the estimated cost for the access plan.

Creating additional indexes on table columns in a single-partition database environment

This example builds on the access plan described in Query 3 by creating an index on the JOB column in the STAFF table, and adding DEPTNAME to the existing index in the ORG table. (Adding a separate index could cause an additional access.)

To view the access plan graph for this query (Query 4): in the Explained Statements History window, double-click the entry identified as Query Number 4. The Access Plan Graph window for this execution of the statement opens.

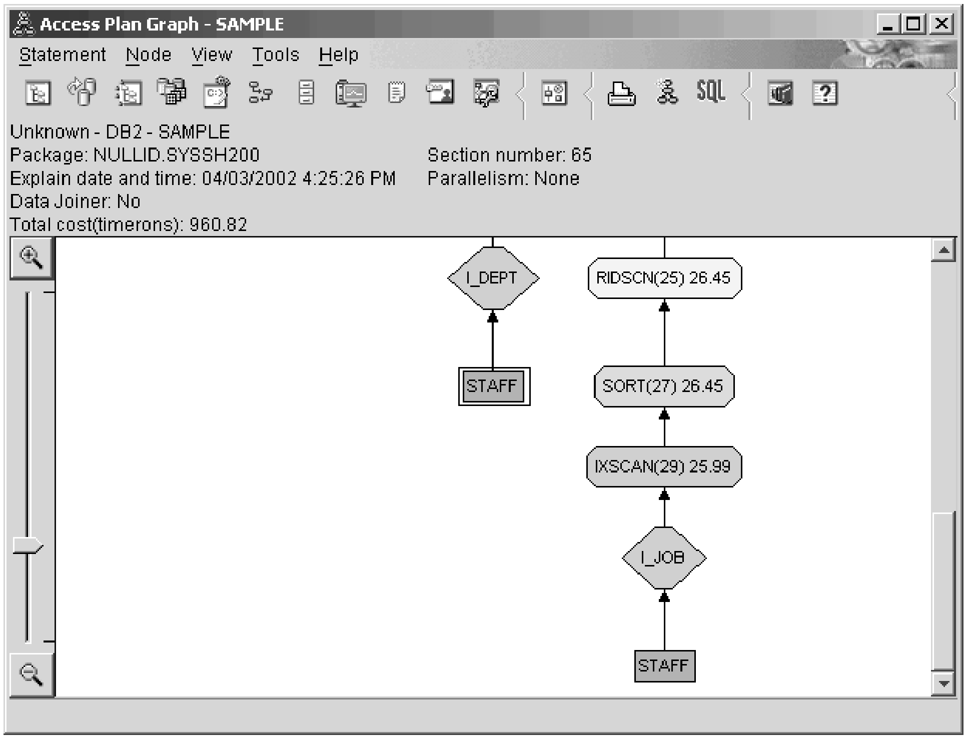


Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 4:25:26 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 960.82

Answering the following questions will help you understand how to improve the query.

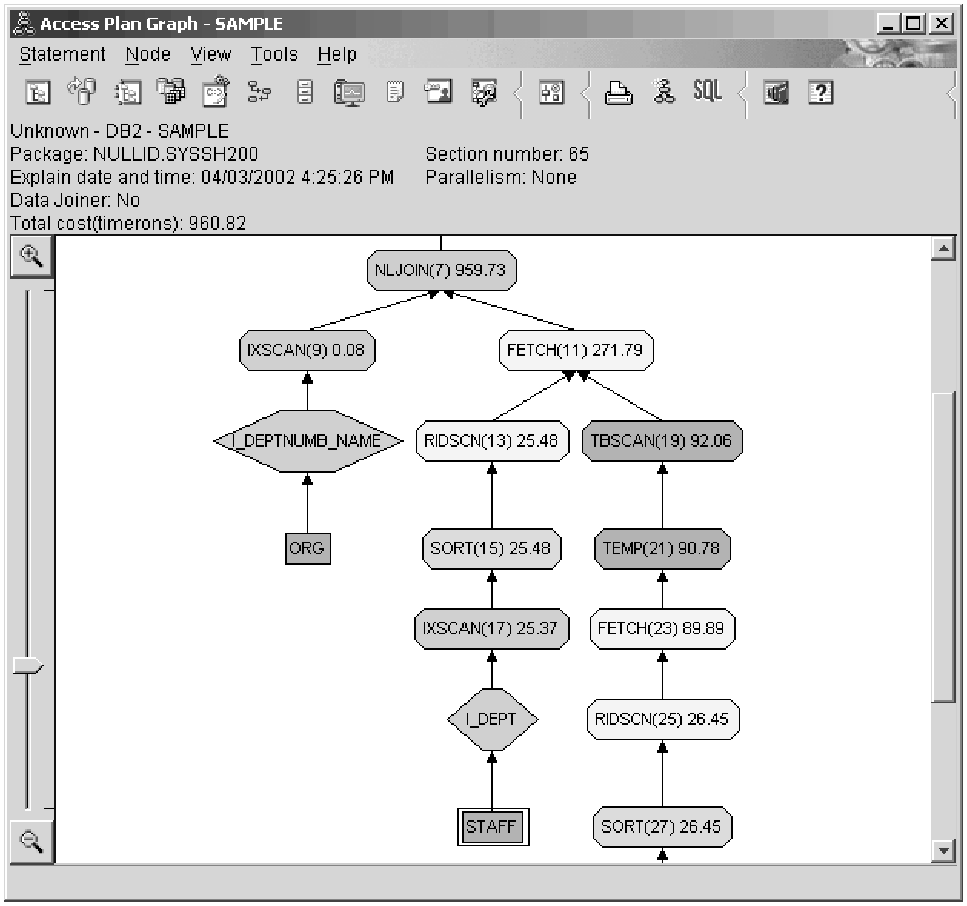
1. What changed in this access plan as a result of creating additional indexes?

   The optimizer has taken advantage of the index created on the JOB column in the STAFF table (represented by a diamond labeled **I_JOB**) to further refine this access plan.



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 4:25:26 PM
Parallelism: None
Data Joiner: No
Total cost(timerons): 960.82

In the middle portion of the access plan graph, notice that for the ORG table, the previous index scan and fetch have been changed to an index scan only IXSCAN (9). Adding the DEPTNAME column to the index on the ORG table has allowed the optimizer to eliminate the extra access involving the fetch.



Unknown - DB2 - SAMPLE
Package: NULLID.SYSSH200 Section number: 65
Explain date and time: 04/03/2002 4:25:26 PM Parallelism: None
Data Joiner: No
Total cost(timerons): 960.82

2. How effective is this access plan?

   This access plan is more cost effective than the one from the previous example. The cumulative cost has been reduced from approximately 1,370 timerons in Query 3 to approximately 959 timerons in Query 4.

What's Next

Improving the performance of your own SQL or XQuery statements.

Refer to the *DB2 Information Center* to find detailed information on additional steps that you can take to improve performance. You can then return to Visual Explain to access the impact of your actions.

Chapter 4. Lesson 4. Improving an access plan in a partitioned database environment

You will learn how the access plan and related windows for the basic query change when you perform various tuning activities.

Using a series of examples, accompanied by illustrations, you will learn how the estimated total cost for the access plan of even a simple query can be improved by using the **runstats** command and adding appropriate indexes.

As you gain experience with Visual Explain, you will discover other ways to tune queries.

Working with access plan graphs

Using the four sample explain snapshots as examples, you will learn how tuning is an important part of database performance.

The queries associated with the explain snapshots are number 1 - 4. Each query uses the same SQL or XQuery statement (described in Lesson 1):

```
SELECT S.ID,SNAME,O.DEPTNAME,SALARY+COMM
FROM ORG O, STAFF S
WHERE
 O.DEPTNUMB = S.DEPT AND
 S.JOB <> 'Mgr' AND
 S.SALARY+S.COMM > ALL ( SELECT ST.SALARY*.9
                         FROM STAFF ST
                         WHERE ST.JOB='Mgr' )
ORDER BY S.NAME
```

But each iteration of the query uses more tuning technics than the previous execution. For example, Query 1 has had no performance tuning, while Query 4 has had the most. The differences in the queries are described below:

Query 1
Running a query with no indexes and no statistics
Query 2
Collecting current statistics for the tables and indexes in a query
Query 3
Creating indexes on columns used to join tables in a query
Query 4
Creating additional indexes on table columns

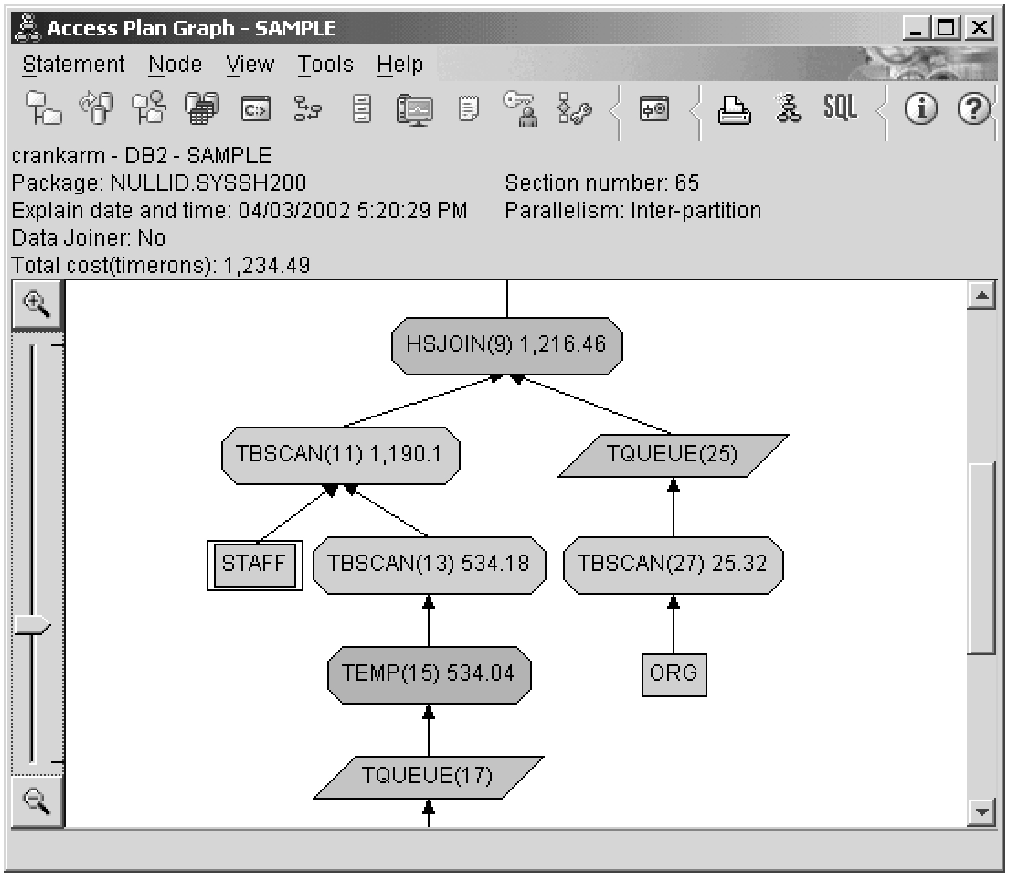
These examples were produced on an RS/6000® SP™ machine with 7 physical nodes using inter-partition parallelism.

Running a query with no indexes and no statistics in a partitioned database environment

In this example, the access plan was created for the SQL query with no indexes and no statistics.

To view the access plan graph for this query (Query 1):

1. In the Control Center, expand the object tree until you find the SAMPLE database.
2. Right-click the database and select **Show explained statements history** from the pop-up menu.
   The Explained Statements History window opens.
3. Double-click the entry identified as Query Number 1 (you might need to scroll to the right to find the **Query Number** column).
   The Access Plan Graph window for the statement opens.



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:20:29 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 1,234.49

Answering the following questions will help you understand how to improve the query:

1. Do current statistics exist for each table in the query?

   To check if current statistics exist for each table in the query, double-click each table node in the access plan graph. In the corresponding Table Statistics window that opens, the **STATS_TIME** row under the **Explained** column contains the words ″Statistics not updated″ indicating that no statistics had been collected at the time when the snapshot was created.

   If current statistics do not exist, the optimizer uses default statistics, which might differ from the actual statistics. Default statistics are identified by the word ″default″ under the **Explained** column in the Table Statistics window.

   According to the information in the Table Statistics window for the ORG table, the optimizer used default statistics (as indicated next to the explained values). Default statistics were used because actual statistics were not available when the snapshot was created (as indicated in the **STATS_TIME** row).

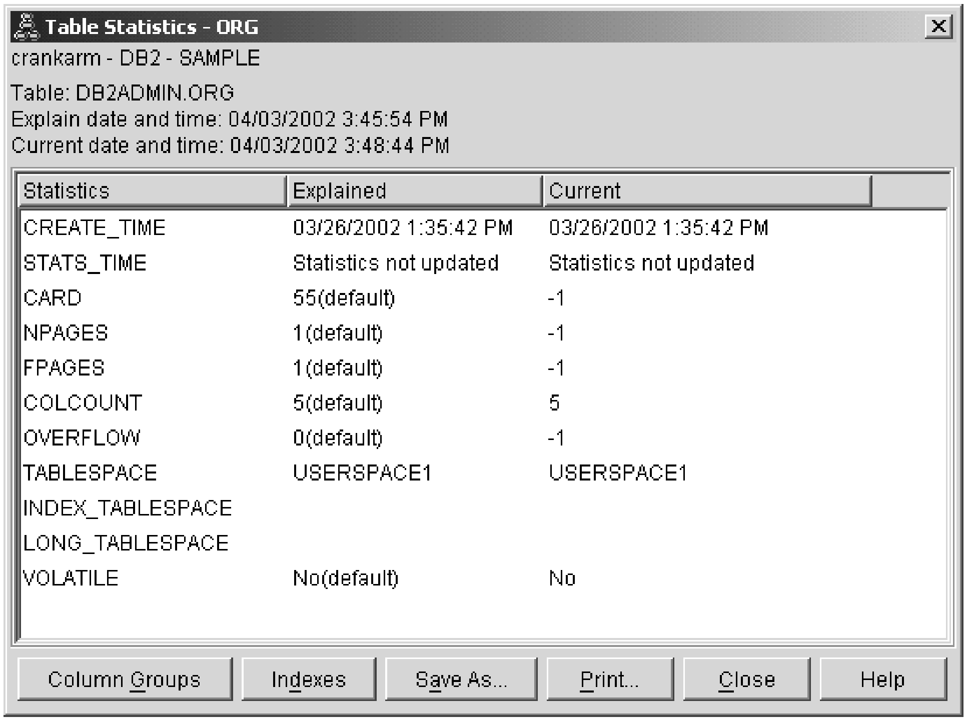


Table Statistics - ORG

crankarm - DB2 - SAMPLE

Table: DB2ADMIN.ORG
Explain date and time: 04/03/2002 3:45:54 PM
Current date and time: 04/03/2002 3:48:44 PM

Statistics	Explained	Current
CREATE_TIME	03/26/2002 1:35:42 PM	03/26/2002 1:35:42 PM
STATS_TIME	Statistics not updated	Statistics not updated
CARD	55(default)	-1
NPAGES	1(default)	-1
FPAGES	1(default)	-1
COLCOUNT	5(default)	5
OVERFLOW	0(default)	-1
TABLESPACE	USERSPACE1	USERSPACE1
INDEX_TABLESPACE		
LONG_TABLESPACE		
VOLATILE	No(default)	No

2. Does this access plan use the most effective methods of accessing data?

   This access plan contains table scans, not index scans. Table scans are shown as octagons and are labeled TBSCAN operator. If Index scans had been used they would appear as diamonds and be labeled IXSCAN. The use of an index that was created for a table is more cost-effective than a table scan if small amounts of data are being extracted.

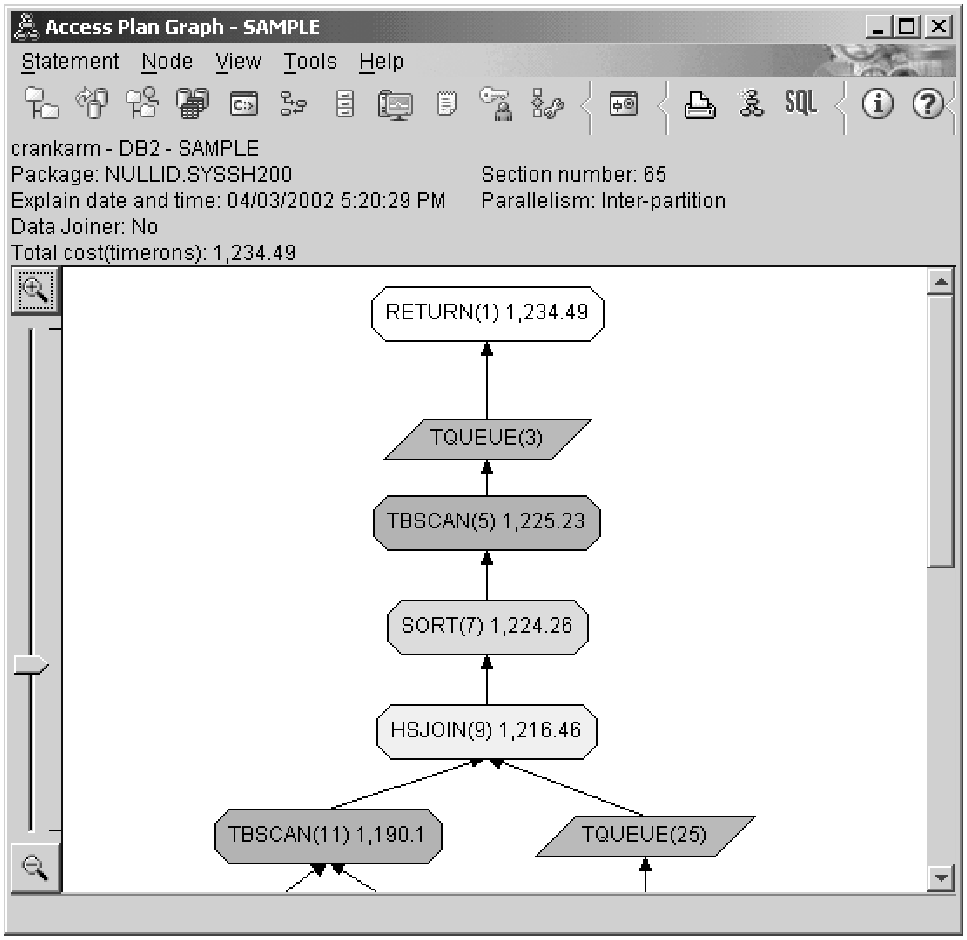
3. How effective is this plan?

   You can determine the effectiveness of an access plan only if it is based on actual statistics. Since the optimizer used default statistics in the access plan, you cannot determine how effective the plan is.

   In general, you should make note of the total estimated Cost for the access plan for later comparison with revised access plans. The cost listed in each node is cumulative, from the first steps of your query up to and including the node.

   Note: For partitioned databases, this is the cumulative cost for the node that uses the most resources.

   In the Access Plan Graph window, the total cost is approximately 1,234 timerons, shown in **RETURN (1)** at the top of the graph. The total estimated cost is also shown in the top area of the window.



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:20:29 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 1,234.49

What's Next

Moving on to query 2.

Query 2 looks at an access plan for the basic query after runstats has been run. Using the runstats command provides the optimizer with current statistics on all tables accessed by the query.

Collecting current statistics for the tables and indexes using runstats in a partitioned database environment

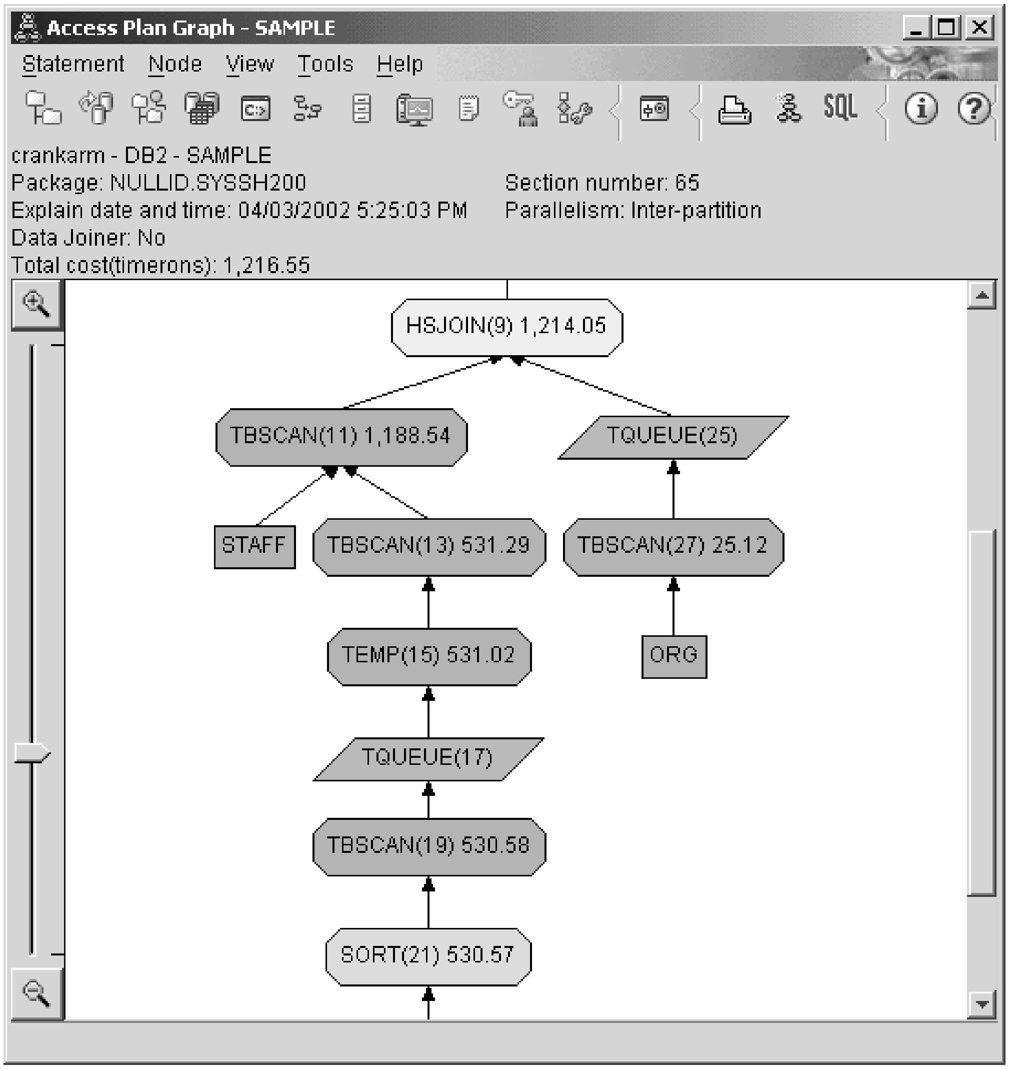
This example builds on the access plan described in Query 1 by collecting current statistics with the runstats command.

It is highly recommended that you use the runstats command to collect current statistics on tables and indexes, especially if significant update activity has occurred or new indexes have been created since the last time the runstats command was executed. This provides the optimizer with the most accurate information with which to determine the best access plan. If current statistics are not available, the optimizer can choose an inefficient access plan based on inaccurate default statistics.

Be sure to use runstats *after* making your table updates; otherwise, the table might appear to the optimizer to be empty. This problem is evident if cardinality on the

Operator Details window equals zero. In this case, complete your table updates, rerun the runstats command, and recreate the explain snapshot for affected tables.

To view the access plan graph for this query (Query 2): in the Explain Statements History window, double-click the entry identified as Query Number 2. The Access Plan Graph window for this execution of the statement opens.



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:25:03 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 1,216.55

Answering the following questions will help you understand how to improve the query.

1. Do current statistics exist for each table in the query?

   The Table Statistics window for the ORG table shows that the optimizer used actual statistics (the STATS_TIME value is the actual time that the statistics were collected). The accuracy of the statistics depends on whether there were significant changes to the contents of the tables since the runstats command was run.

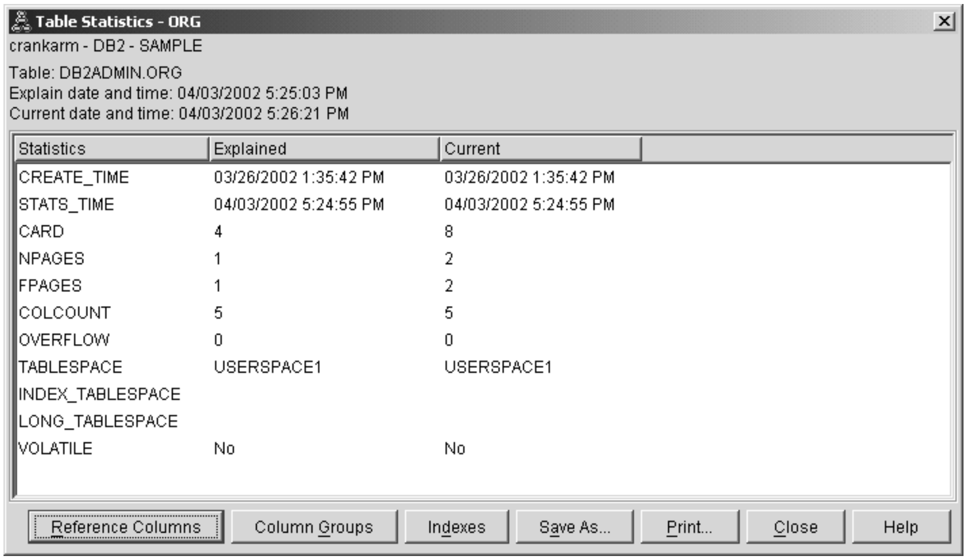


Table Statistics - ORG

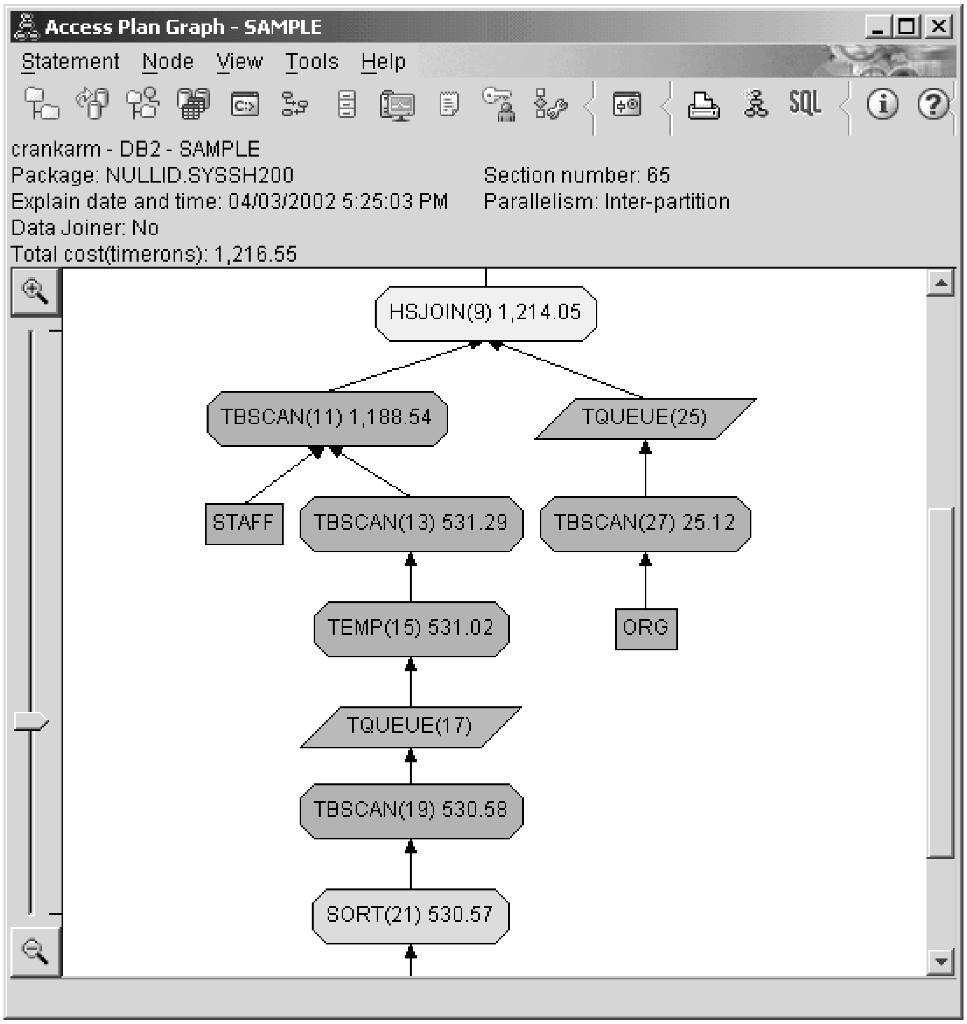
crankarm - DB2 - SAMPLE

Table: DB2ADMIN.ORG
Explain date and time: 04/03/2002 5:25:03 PM
Current date and time: 04/03/2002 5:26:21 PM

Statistics	Explained	Current
CREATE_TIME	03/26/2002 1:35:42 PM	03/26/2002 1:35:42 PM
STATS_TIME	04/03/2002 5:24:55 PM	04/03/2002 5:24:55 PM
CARD	4	8
NPAGES	1	2
FPAGES	1	2
COLCOUNT	5	5
OVERFLOW	0	0
TABLESPACE	USERSPACE1	USERSPACE1
INDEX_TABLESPACE		
LONG_TABLESPACE		
VOLATILE	No	No

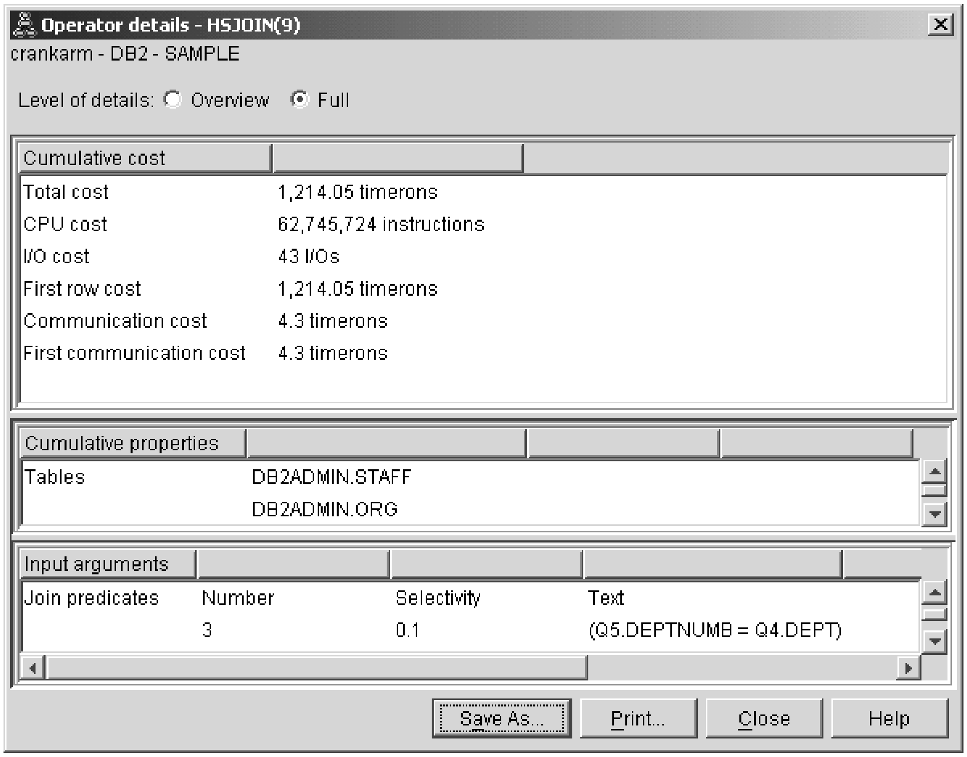
2. Does this access plan use the most effective methods of accessing data?

   Like Query 1, the access plan in Query 2 uses table scans (TBSCAN operator) not index scans (IXSCAN). Even though current statistics exist, an index scan was not done because there are no indexes on the columns that were used by the query. One way to improve the query would be to provide the optimizer with indexes on columns that are used to join tables (that is, on columns that are used in join Predicates). In this example, this is the first merge scan join: HSJOIN (9).



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:25:03 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 1,216.55

In the Operator Details window for the HSJOIN (9) operator, look at the **Join predicates** section under **Input arguments**. The columns used in this join operation are listed under the **Text** column. in this example, these columns are DEPTNUMB and DEPT.



Operator details - HSJOIN(9)

crankarm - DB2 - SAMPLE

Level of details: ○ Overview ● Full

Cumulative cost	
Total cost	1,214.05 timerons
CPU cost	62,745,724 instructions
I/O cost	43 I/Os
First row cost	1,214.05 timerons
Communication cost	4.3 timerons
First communication cost	4.3 timerons

Cumulative properties	
Tables	DB2ADMIN.STAFF
	DB2ADMIN.ORG

Input arguments			
Join predicates	Number	Selectivity	Text
	3	0.1	(Q5.DEPTNUMB = Q4.DEPT)

3. How effective is this access plan?

   Access plans based on up-to-date statistics always produce a realistic estimated cost (measured in timerons). Because the estimated cost in Query 1 was based on default statistics, the cost of the two access plan graphs cannot be compared to determine which one is more effective. Whether the cost is higher or lower is not relevant. You must compare the cost of access plans that are based on actual statistics to get a valid measurement of effectiveness.

What's Next

Moving on to query 3.

Query 3 looks at the effects of adding indexes on the DEPTNUMB and DEPT columns. Adding indexes on the columns that are used in join predicates can improve performance.

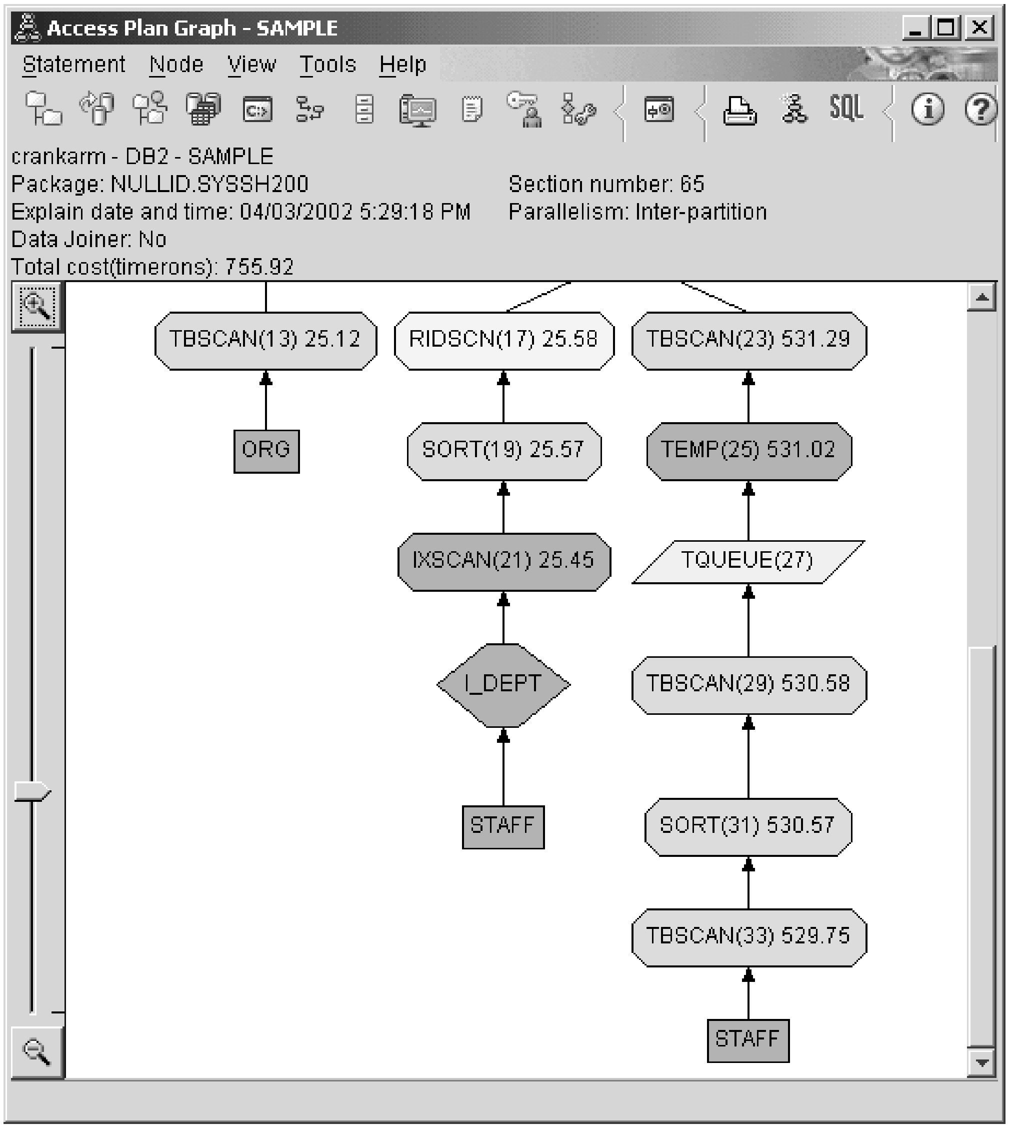
Creating indexes on columns used to join tables in a query in a partitioned database environment

This example builds on the access plan described in Query 2 by creating indexes on the DEPT column on the STAFF table and on the DEPTNUMB column on the ORG table.

Note: Recommended indexes can be created using the Design Advisor.

To view the access plan graph for this query (Query 3): in the Explained Statements History window, double-click the entry identified as Query Number 3. The Access Plan Graph window for this execution of the statement opens.

Note: Even though an index was created for DEPTNUM, the optimizer did not use it.



crankarm - DB2 - SAMPLE

Package: NULLID.SYSSH200

Section number: 65

Explain date and time: 04/03/2002 5:29:18 PM

Parallelism: Inter-partition

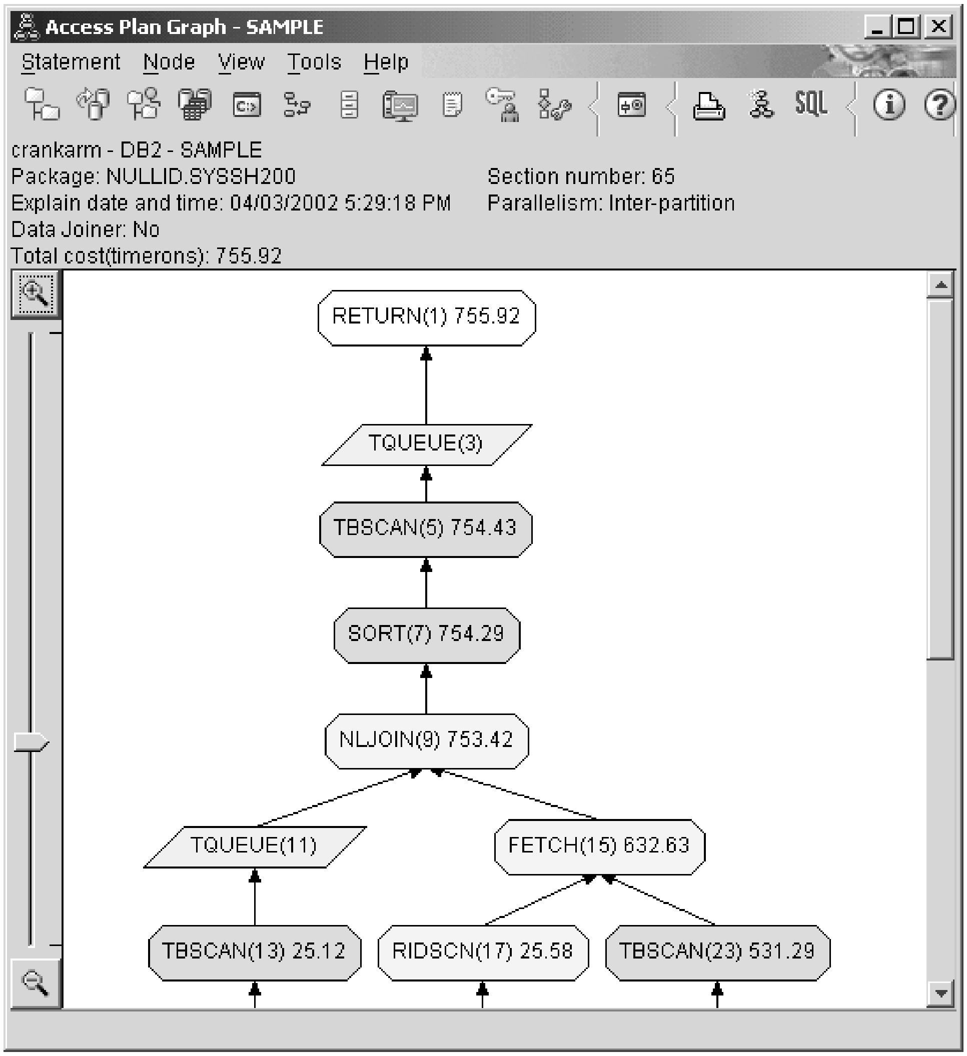
Data Joiner: No

Total cost(timerons): 755.92

Answering the following questions will help you understand how to improve the query.

1. What has changed in the access plan with indexes?

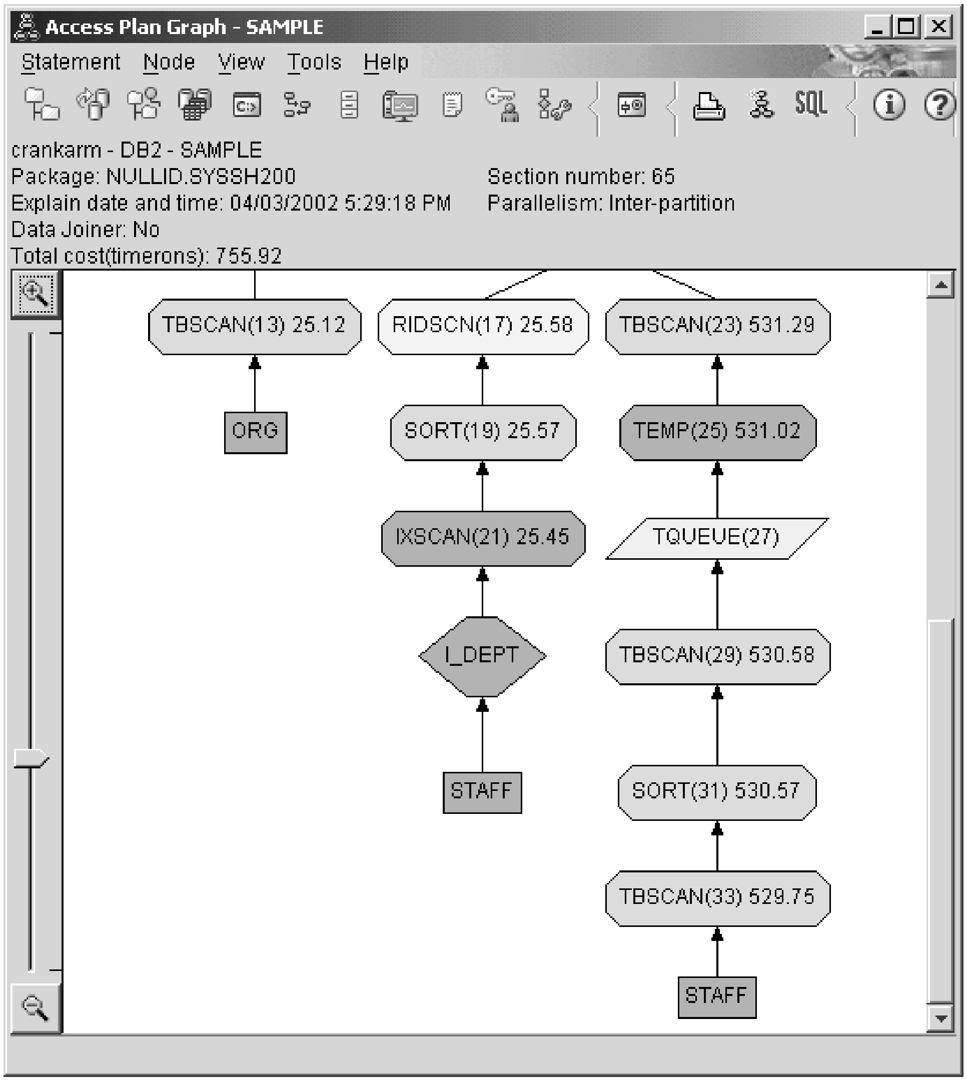
   A new diamond-shaped node, **I_DEPT**, has been added just above the STAFF table. This node represents the index that was created on DEPT, and it shows that the optimizer used an index scan instead of a table scan to determine which rows to retrieve.



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200 Section number: 65
Explain date and time: 04/03/2002 5:29:18 PM Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 755.92

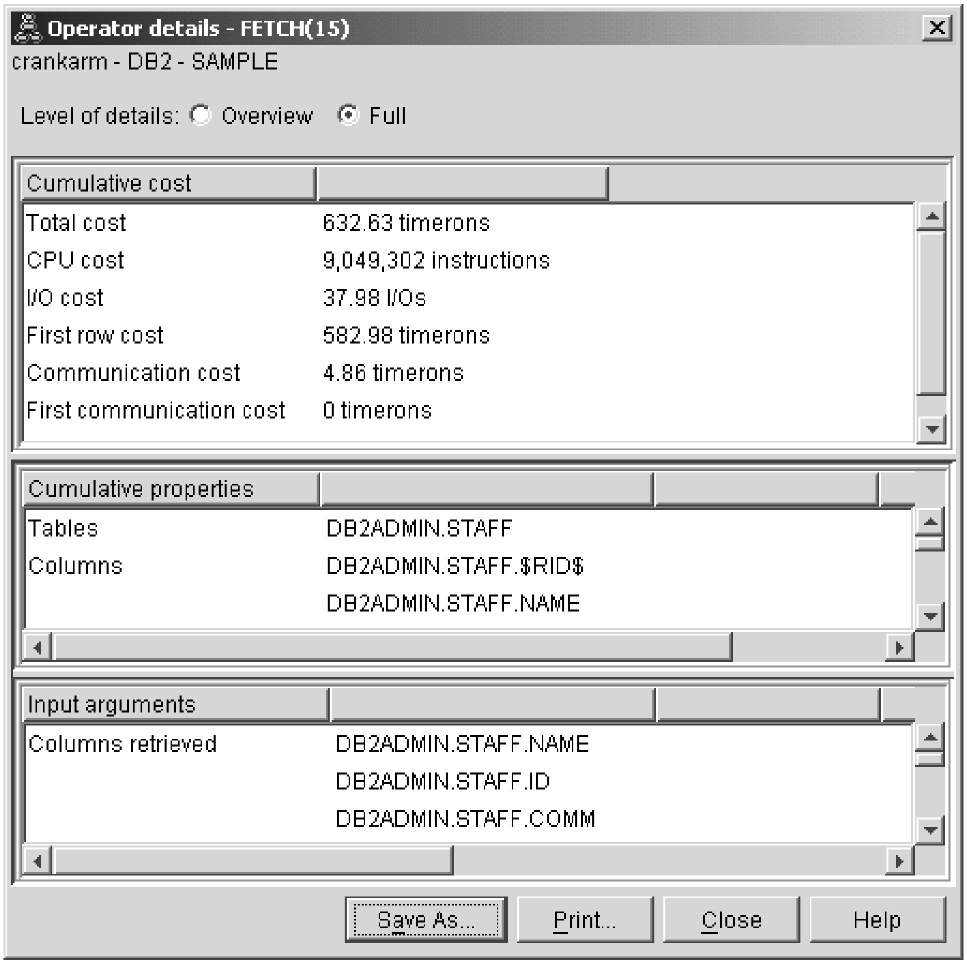
2. Does this access plan use the most effective methods of accessing data?

   The access plan for this query shows the effect of creating indexes on the DEPTNUMB column of the ORG table, resulting in FETCH (15) and IXSCAN (21) and on the DEPT column of the STAFF table. Query 2 did not have this index; therefore, a table scan was used in that example.



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:29:18 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 755.92

The Operator Details window for the FETCH (15) operator shows the columns being used in this operation.



Operator details - FETCH(15)

crankarm - DB2 - SAMPLE

Level of details: ○ Overview ● Full

Cumulative cost	
Total cost	632.63 timerons
CPU cost	9,049,302 instructions
I/O cost	37.98 I/Os
First row cost	582.98 timerons
Communication cost	4.86 timerons
First communication cost	0 timerons

Cumulative properties	
Tables	DB2ADMIN.STAFF
Columns	DB2ADMIN.STAFF.RID
	DB2ADMIN.STAFF.NAME

Input arguments	
Columns retrieved	DB2ADMIN.STAFF.NAME
	DB2ADMIN.STAFF.ID
	DB2ADMIN.STAFF.COMM

The combination of index and fetch are calculated to be less costly than the full table scans used in the previous access plans.

3. How effective is this access plan?

   This access plan is more effective than the one from the previous example. The cumulative cost has been reduced from approximately 1,214 timerons in Query 2 to approximately 755 timerons in Query 3.

What’s Next

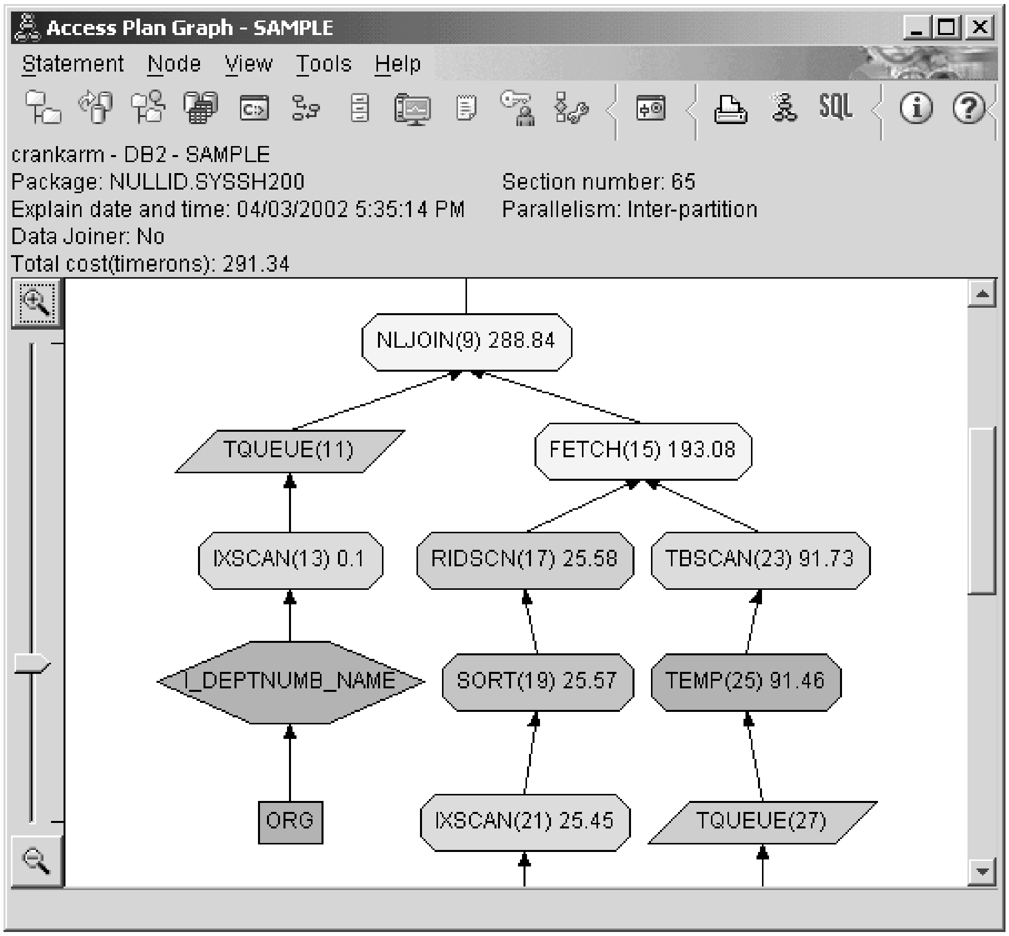
Moving on to query 4.

Query 4 reduces the fetch and index scan to a single index scan without a fetch. Creating additional indexes can reduce the estimated cost for the access plan.

Creating additional indexes on table columns in a partitioned database environment

This example builds on the access plan described in Query 3 by creating an index on the JOB column in the STAFF table, and adding DEPTNAME to the existing index in the ORG table. (Adding a separate index could cause an additional access.)

To view the access plan graph for this query (Query 4): in the Explained Statements History window, double-click the entry identified as Query Number 4. The Access Plan Graph window for this execution of the statement opens.

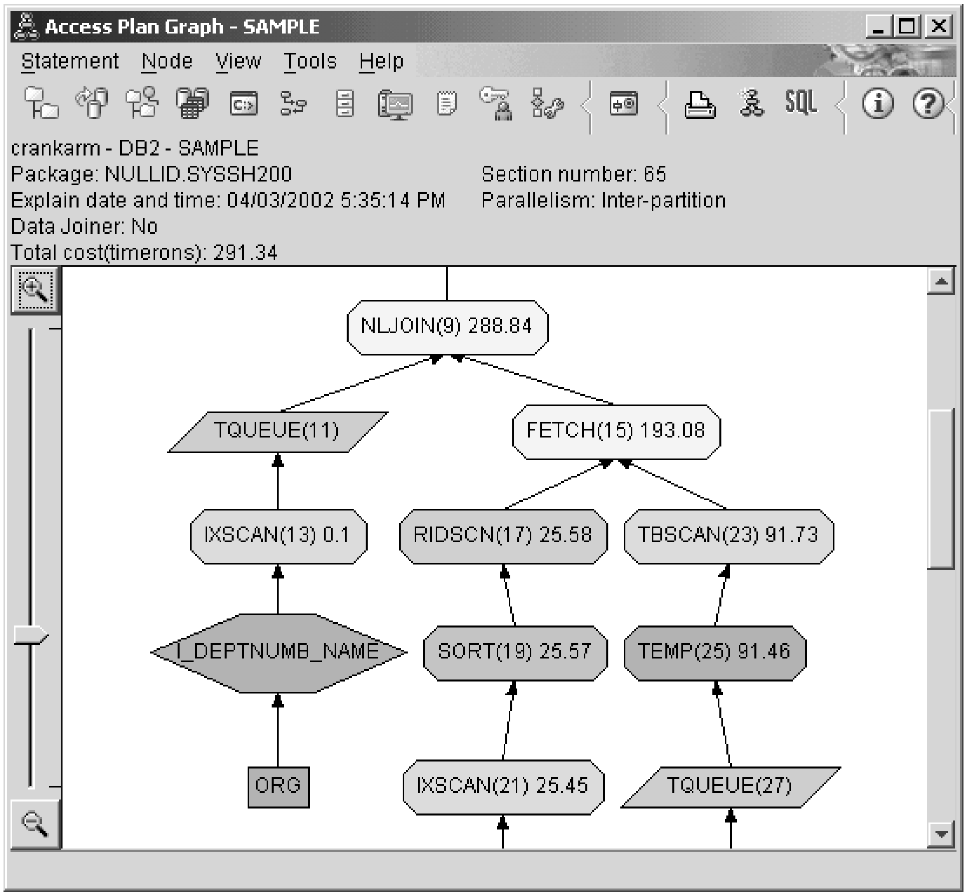


crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:35:14 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 291.34

Answering the following questions will help you understand how to improve the query.

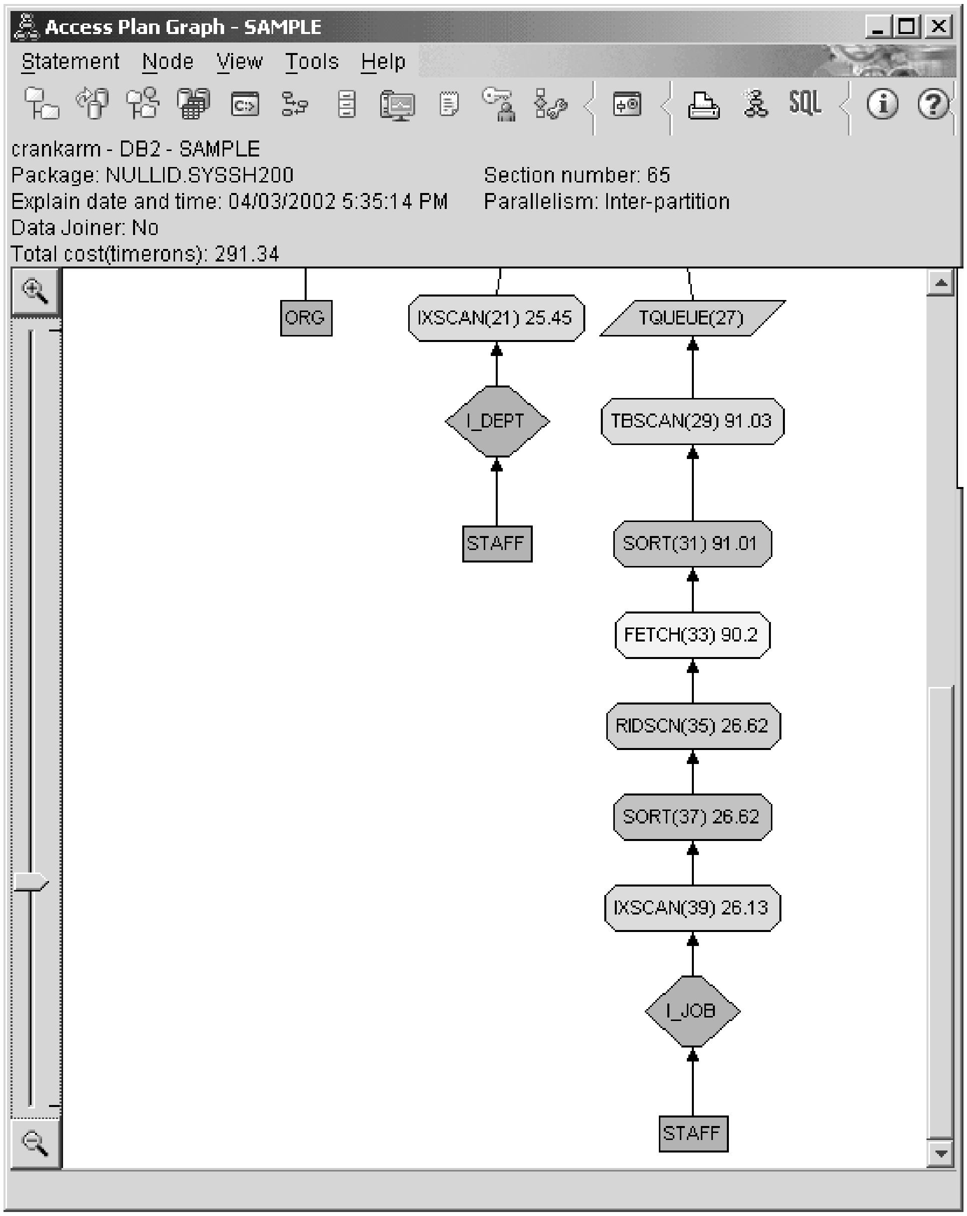
1. What changes in this process plan as a result of creating additional indexes?

   In the middle portion of the access plan graph, notice that for the ORG table, the previous table scan has been changed to an index scan, IXSCAN (7). Adding the DEPTNAME column to the index on the ORG table has allowed the optimizer to refine the access involving the table scan.



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:35:14 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 291.34

In the bottom portion of the access plan graph, note that for the STAFF table the previous index scan and fetch have been changed to an index scan only IXSCAN (39). Creating the JOB index on the STAFF table has allowed the optimizer to eliminate the extra access involving the fetch.



crankarm - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 04/03/2002 5:35:14 PM
Parallelism: Inter-partition
Data Joiner: No
Total cost(timerons): 291.34

2. How effective is this access plan?

   This access plan is more cost effective than the one from the previous example. The cumulative cost has been reduced from approximately 753 timerons in Query 3 to approximately 288 timerons in Query 4.

What's Next

Improving the performance of your own SQL or XQuery statements.

Refer to the *DB2 Information Center* to find detailed information on additional steps that you can take to improve performance. You can then return to Visual Explain to access the impact of your actions.

Part 2. Reference

Chapter 5. Visual Explain operators

An *operator* is either an action that must be performed on data, or the output from a table or an index, when the access plan for an SQL or XQuery statement is executed. This section contains a list of the operators that can appear in an access plan graph.

CMPEXP operator

This operator is for debug mode only.

Operator name: CMPEXP

Represents: The computation of expressions required for intermediate or final results.

DELETE operator

This operator represents the deletion of rows from a table.

Operator name: DELETE

Represents: The deletion of rows from a table.

This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be deleted.

Performance Suggestion:

- If you are deleting all rows from a table, consider using the DROP TABLE statement or the LOAD REPLACE command.

EISCAN operator

This operator scans a user defined index to produce a reduced stream of rows.

Operator name: EISCAN

Represents: The scanning uses the multiple start/stop conditions from the user supplied range producer function.

This operation is performed to narrow down the set of qualifying rows before accessing the base table (based on predicates).

Performance Suggestion:

- Over time, database updates can cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or reorganizing the index.
- If statistics are not current, update them using the runstats command.

FETCH

This operator represents the fetching of columns from a table.

Operator name: FETCH

Represents: The fetching of columns from a table using a specific row identifier (RID).

Performance suggestions:

- Expand index keys to include the fetched columns so that the data pages do not have to be accessed.
- Find the index related to the fetch, and double-click on its node to display its statistics window. Ensure that the degree of clustering is high for the index.
- Increase the buffer size if the input/output (I/O) incurred by the fetch is greater than the number of pages in the table.
- If statistics are not current, update them.

  The quantile and frequent value statistics provide information on the selectivity of predicates, which determines when index scans are chosen over table scans. To update these statistics, use the runstats command on a table with the WITH DISTRIBUTION clause.

FILTER operator

This operator represents how data is filtered.

Operator name: FILTER

Represents: The application of residual predicates so that data is filtered based on the criteria supplied by the predicates.

Performance suggestions:

- Ensure that you have used predicates that retrieve only the data you need. For example, ensure that the selectivity value for the predicates represents the portion of the table that you want returned.
- Ensure that the optimization class is at least 3 so that the optimizer uses a join instead of a subquery. If this is not possible, try rewriting the SQL query by hand to eliminate the subquery.

GENROW operator

This operator is used by the optimizer to generate rows of data.

Operator name: GENROW

Represents: A built-in function that generates a table of rows, using no input from tables, indexes, or operators.

GENROW can be used by the optimizer to generate rows of data (for example, for an INSERT statement or for some IN-lists that are transformed into joins).

To view the estimated statistics for the tables generated by the GENROW function, double-click on its node.

GRPBY operator

This operator represents the grouping of rows.

Operator name: GRPBY

Represents: The grouping of rows according to common values of designated columns or functions. This operation is required to produce a group of values, or to evaluate set functions.

If no GROUP BY columns are specified, the GRPBY operator can still be used if there are aggregation functions in the SELECT list, indicating that the entire table is treated as a single group when doing that aggregation.

Performance suggestions:

- This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be grouped.
- To improve the performance of a SELECT statement that contains a single aggregate function but no GROUP BY clause, try the following:
  - For a MIN(C) aggregate function, create an ascending index on C.
  - For a MAX(C) aggregate function, create a descending index on C.

HSJOIN operator

This operator represents hash joins for which the qualified rows from tables are hashed.

Operator name: HSJOIN

Represents: A hash join for which the qualified rows from tables are hashed to allow direct joining, without pre-ordering the content of the tables.

A join is necessary whenever there is more than one table referenced in a FROM clause. A hash join is possible whenever there is a join predicate that equates columns from two different tables. The join predicates need to be exactly the same data type. Hash joins can also arise from a rewritten subquery, as is the case with NLJOIN .

A hash join does not require the input tables be ordered. The join is performed by scanning the inner table of the hash join and generating a lookup table by hashing the join column values. It then reads the outer table, hashing the join column values, and checking in the lookup table generated for the inner table.

Performance suggestions:

- Use local predicates (that is, predicates that reference one table) to reduce the number of rows to be joined.
- Increase the size of the sort heap to make it large enough to hold the hash lookup table in memory.
- If statistics are not current, update them using the runstats command.

INSERT operator

This operator represents the insertion of rows into a table.

Operator name: INSERT

Represents: This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be inserted.

IXAND operator

This operator represents the ANDing of the results of multiple index scans.

Operator name: IXAND

Represents: The ANDing of the results of multiple index scans using Dynamic Bitmap techniques. The operator allows ANDed predicates to be applied to multiple indexes, in order to reduce underlying table accesses to a minimum.

This operator is performed to:

- Narrow down the set of rows before accessing the base table
- AND together predicates applied to multiple indexes
- AND together the results of semijoins, used in star joins.

Performance suggestions:

- Over time, database updates can cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or reorganizing the index.
- If statistics are not current, update them using the runstats command .
- In general, index scans are most effective when only a few rows qualify. To estimate the number of qualifying rows, the optimizer uses the statistics that are available for the columns referenced in predicates. If some values occur more frequently than others, it is important to request distribution statistics by using the WITH DISTRIBUTION clause for the runstats command. By using the non-uniform distribution statistics, the optimizer can distinguish among frequently and infrequently occurring values.
- IXAND can best exploit single column indexes, as start and stop keys are critical in the use of IXAND.
- For star joins, create single-column indexes for each of the most selective columns in the fact table and the related dimension tables.

IXSCAN

This operator represents the scanning of an index.

Operator name: IXSCAN

Represents: The scanning of an index to produce a reduced stream of row IDs. The scanning can use optional start/stop conditions, or might apply to indexable predicates that reference columns of the index.

This operation is performed to narrow down the set of qualifying row IDs before accessing the base table (based on predicates).

Performance suggestions:

- Over time, database updates can cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or reorganizing the index.
- When two or more tables are being accessed, access to the inner table via an index can be made more efficient by providing an index on the join column of the outer table.

  For more guidelines about indexes, see the online help for Visual Explain.
- If statistics are not current, update them using the runstats command.
- In general, index scans are most effective when only a few row IDs qualify. To estimate the number of qualifying row IDs, the optimizer uses the statistics that are available for the columns referenced in predicates. If some values occur more frequently than others, it is important to request distribution statistics by using the WITH DISTRIBUTION clause for the runstats command. By using the non-uniform distribution statistics, the optimizer can distinguish among frequently and infrequently occurring values.

MSJOIN operator

This operator represents a merge join.

Operator name: MSJOIN

Represents: A merge join for which the qualified rows from both outer and inner tables must be in join-predicate order. A merge join is also called a *merge scan join* or a *sorted merge join* .

A join is necessary whenever there is more than one table referenced in a FROM clause. A merge join is possible whenever there is a join predicate that equates columns from two different tables. It can also arise from a rewritten subquery.

A merge join requires ordered input on joining columns, since the tables are typically scanned only once. This ordered input is obtained by accessing an index or a sorted table.

Performance suggestions:

- Use local predicates (that is, predicates that reference one table) to reduce the number of rows to be joined.
- If statistics are not current, update them using the runstats command.

NLJOIN operator

This operator represents a nested loop join.

Operator name: NLJOIN

Represents: A nested loop join that scans (usually with an index scan) the inner table once for each row of the outer table.

A join is necessary whenever there is more than one table referenced in a FROM clause. A nested loop join does not require a join predicate, but generally performs better with one.

A nested loop join is performed either:

- By scanning through the inner table for each accessed row of the outer table.
- By performing an index lookup on the inner table for each accessed row of the outer table.

Performance suggestions:

- A nested loop join is likely to be more efficient if there is an index on the join-predicate columns of the inner table (the table displayed to the right of the NLJOIN operator). Check to see if the inner table is a TBSCAN rather than an IXSCAN. If it is, consider adding an index on its join columns.

  Another (less important) way to make the join more efficient is to create an index on the join columns of the outer table so that the outer table is ordered.
- If statistics are not current, update them using the runstats command.

PIPE operator

This operator is for debug mode only.

Operator name: PIPE

Represents: The transfer of rows to other operators without any change to the rows.

RETURN operator

This operator represents the return of data from a query.

Operator name: RETURN

Represents: The return of data from a query to the user. This is the final operator in the access plan graph and shows the total accumulated values and costs for the access plan.

This operator represents a necessary operation.

Performance Suggestion:

- Ensure that you have used predicates that retrieve only the data you need. For example, ensure that the selectivity value for the predicates represents the portion of the table that you want returned.

RIDSCN operator

This operator represents the scan of a list of row identifiers (RIDs).

Operator name: RIDSCN

Represents: The scan of a list of row identifiers (RIDs) obtained from one or more indexes.

This operator is considered by the optimizer when:

- Predicates are connected by OR keywords, or there is an IN predicate. A technique called index ORing can be used, which combines results from multiple index accesses on the same table.
- It is beneficial to use list prefetch for a single index access, since sorting the row identifiers before accessing the base rows makes the I/O more efficient.

RPD operator

This operator retrieves data from a remote data source.

Operator name: RPD

Represents: An operator used in the federated system to retrieve data from a remote data source via a non-relational wrapper.

This operator is considered by the optimizer when it contains a remote plan that will not be inspected by the optimizer. An RPD operator sends a request to a remote non-relational data source to retrieve the query result. The request is generated by the non-relational wrapper using the API supported by the data source.

SHIP operator

This operator retrieves data from a remote data source.

Operator name: SHIP

Represents: An operator used in the federated system to retrieve data from a remote data source. This operator is considered by the optimizer when it contains a remote plan that will not be inspected by the optimizer. A SHIP operator sends an SQL or XQuery SELECT statement to a remote data source to retrieve the query result. The SELECT statement is generated using the SQL or XQuery dialect supported by the data source, and can contain any valid query as allowed by the data source.

SORT operator

This operator represents the sorting of rows in a table.

Operator name: SORT

Represents: The sorting of the rows in a table into the order of one or more of its columns, optionally eliminating duplicate entries.

Sorting is required when no index exists that satisfies the requested ordering, or when sorting would be less expensive than an index scan. Sorting is usually performed as a final operation once the required rows are fetched, or to sort data prior to a join or a group by.

If the number of rows is high or if the sorted data cannot be piped, the operation requires the costly generation of temporary tables.

Performance suggestions:

- Consider adding an index on the sort columns.
- Ensure that you have used predicates that retrieve only the data you need. For example, ensure that the selectivity value for the predicates represents the portion of the table that you want returned.
- Check that the prefetch size of the system temporary table space is adequate, that is, it is not I/O bound. (To check this, select **Statement–>Show Statistics–>Table Spaces**.)

- If frequent large sorts are required, consider increasing the values of the following configuration parameters:
  - Sort heap size (sortheap). To change this parameter, right-click on the database in the Control Center, and then select **Configure** from its pop-up menu. Select the Performance tab from the notebook that opens.
  - Sort heap threshold (sheapthres). To change this parameter, right-click on the database instance in the Control Center, and then select **Configure** from its pop-up menu. Select the Performance tab from the notebook that opens.
- If statistics are not current, update them using the runstats command.

TBSCAN operator

This operator represents table scans.

Operator name: TBSCAN

Represents: A table scan (relation scan) that retrieves rows by reading all the required data directly from the data pages.

This type of scan is chosen by the optimizer over an index scan when:
- The range of values scanned occurs frequently (that is, most of the table must be accessed)
- The table is small
- Index clustering is low
- An index does not exist

Performance suggestions:
- An index scan is more efficient than a table scan if the table is large, with most of the table's rows not being accessed. To increase the possibility that an index scan will be used by the optimizer for this situation, consider adding indexes on columns for which there are selective predicates.
- If an index already exists but was not used, check that there are selective predicates on each of its leading columns. If these predicates do exist, next check that the degree of clustering is high for the index. (To see this statistic, open the Table Statistics window for the table beneath the sort, and select its *Indexes* push button to bring up the Index Statistics window.)
- Check that the prefetch size of the table space is adequate that is, it is not I/O bound. (To check this, select **Statement–>Show Statistics–>Table Spaces**.)
- If the statistics are not current, update them using the runstats command.

  The quantile and frequent value statistics provide information on the selectivity of predicates. For example, these statistics would be used to determine when index scans are chosen over table scans. To update these values, use the runstats command on a table with the WITH DISTRIBUTION clause.

TEMP operator

This operator represents the storage of data in a temporary table.

Operator name: TEMP

Represents: The action of storing data in a temporary table, to be read back out by another operator (possibly multiple times). The table is removed after the SQL or XQuery statement is processed, if not before.

This operator is required to evaluate subqueries or to store intermediate results. In some situations (such as when the statement can be updated), it might be mandatory.

TQ operator

This operator represents a table queue.

Operator name: TQ

Represents: A table queue that is used to pass table data from one database agent to another when there are multiple database agents processing a query. Multiple database agents are used to process a query when parallelism is involved. The table queue types are:

- **Local**: The table queue is used to pass data between database agents within a single node. A local table queue is used for intrapartition parallelism.
- **Non-Local**: The table queue is used to pass data between database agents on different nodes.

There are two types of TQ operators:

- ATQ - Asynchronous TQ operator
- XTQ - XML aggregation TQ Operator

The ATQ operator enables asynchronous execution of a subplan.

The XTQ operator is a table queue that constructs a XML sequence from XML documents stored on database partitions.

In the following example, US_ORDERS is a partitioned table that is spread across several database partitions, and US_ORDERS.DETAILS is an XML column. The following XQuery statement returns all orders if the total number of products sold exceeds 100:

```
Xquery let $all_orders := db2-fn:xmlcolumn('US_ORDERS.DETAILS')
```

```
Xquery let $all_orders := db2-fn:xmlcolumn('US_ORDERS.DETAILS')
```

where

```
sum($all_orders//product/qty) > 100
```

return

```
$all_orders
```

For the XQuery statement, the db2exfmt command produces the following access plan output that contains an XTQ operator. In the plan output, references to all the XML documents stored in US_ORDERS.DETAILS column are routed to the coordinator partition and aggregated into a global sequence, later each item in the global sequence is routed back to its original partition for navigation and the results are aggregated together into a new output global sequence.

```
 Rows
RETURN
(   1)
 Cost
 I/O
  |
  1
NLJOIN
```

```
          (   2)
           98.171
             8
           /-+-\
       1         2000
    FILTER        XTQ
    (   3)       (   7)
    37.4289      60.7421
       1           7
       |           |
       1          0.5
    GRPBY        XSCAN
    (   4)       (   8)
    37.3755       57.2
       1            7
       |
      360
      DTQ
    (   5)
    33.7655
       1
       |
      180
    TBSCAN
    (   6)
    22.051
       1
       |
      180
TABLE: USER1
   US_ORDERS
```

UNION operator

This operator represents the concatenation of steams of rows from multiple tables.

Operator name: UNION

Represents: This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be concatenated.

UNIQUE operator

This operator represents rows with duplicate values.

Operator name: UNIQUE

Represents: The elimination of rows having duplicate values for specified columns.

Performance Suggestion:

- This operator is not necessary only if a unique index exists on appropriate columns.

  For guidelines about indexes, see Creating appropriate indexes in the online help for Visual Explain.

UPDATE operator

This operator represents the updating of data in the rows of a table.

Operator name: UPDATE

Represents: This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be updated.

XANDOR operator

This operator allows ANDed predicates to be applied to multiple indexes to reduce underlying table accesses to a minimum.

Operator name: XANDOR

Represents: The index over XML data ANDing of the results of multiple index scans, used for the evaluation of complex predicates from a single query.

In order for the XANDOR operator to be used, the following conditions must be met:

- Only equality predicates are used.
- There are no wildcards in the index lookup path.
- All predicates are used on the same XML column.

If any of these conditions are not met the IXAND operator will be used instead.

An access plan with multiple XANDORed index over XML data scans as shown by the db2exfmt tool might look like this:

```
            Rows
           RETURN
           (   1)
            Cost
             I/O
             |
         0.00915933
           NLJOIN
           (   2)
           985.789
           98.9779
           /--+--\
      2.96215  0.00309213
      FETCH      XSCAN
      (   3)     (  11)
      340.113    217.976
        19         27
     /---+---\
2.96215      210000
RIDSCN   TABLE: DB2XML
(   4)        TPCHX
332.008
  18
  |
2.96215
SORT
(   5)
331.957
  18
  |
                             2.96215
                             XANDOR
                             (   6)
                             331.784
                               18
      +----------------+--------+-------+----------------+
    355.62           6996.81          105000           105000
    XISCAN           XISCAN           XISCAN           XISCAN
    (   7)           (   8)           (   9)           (  10)
    165.892          3017.54       1.6473e+06          851554
       9               81              27768            14898
      |                |                |                |
    210000           210000           210000           210000
XMLIN: DB2XML    XMLIN: DB2XML    XMLIN: DB2XML    XMLIN: DB2XML
   TPCHX_IDX        TPCHX_IDX        TPCHX_IDX        TPCHX_IDX
```

Each XISCAN operator will perform an index scan and feed the XANDOR operator with the XML node IDs that qualify. The XANDOR operator will apply the AND and OR predicates and return the XML nodes that satisfy the XML pattern for the query.

Performance suggestions:

- Over time, database updates can cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or by reorganizing it.
- If statistics are not current, update them using the RUNSTATS command.
- In general, index scans are most effective when only a few rows qualify. To estimate the number of qualifying rows, the optimizer uses the statistics that are available for the columns referenced in predicates. If some values occur more frequently than others, it is important to request distribution statistics by using the WITH DISTRIBUTION clause with the RUNSTATS command. By using the non-uniform distribution statistics, the optimizer can distinguish among frequently and infrequently occurring values.

XISCAN operator

This operation is performed for a single query predicate.

Operator name: XISCAN

Represents: Its evaluation narrows down the qualifying return set of rows IDs and XML node IDs by range scanning any associated index over XML data before accessing the base table. The use of an index can improve the performance of a query because the compiler determines whether and how to use index information to complete the query. This typically results in:

- The sorting of nodes by document and the elimination of any duplicates.
- The fetching of each row of the table that contains the qualifying documents.
- An XSCAN operation on the XML document.

For example, if you want to find the first name of all of the people listed in the sample XML document fragments who have the last name ″Murphy″, a valid XQuery statement to fetch the correct results is:

```
db2-fn:column("EMPLOYEE.XMLCOL")/emp//name[last="Murphy"]/first
```

Assume that you had previously created an index over XML data on all last names by using the following statement:

```
CREATE INDEX empname on EMPLOYEE(XMLCOL)
   GENERATE KEY USING XMLPATTERN '//name/last'
   AS SQL VARCHAR(50)
```

The query compiler can choose an XISCAN operator to evaluate the query, which will receive the pattern: /emp//name/last, the operator ″=″, and the value ″Murphy″. The index will help to quickly locate the nodes associated with the last name ″Murphy″. The resulting plan fragment as output by `db2exfmt` might look like this:

```
                Rows
               RETURN
               (   1)
                Cost
                 I/O
                 |
               6454.4
               NLJOIN
               (   2)
             1.53351e+06
               189180
               /--+-\
          6996.81  0.922477
          FETCH     XSCAN
          (   3)    (   7)
          4091.76   218.587
            266       27
         /---+---\
    6996.81      210000
    RIDSCN   TABLE: DB2XML
    (   4)        TX
    3609.39
      81
      |
    6996.81
    SORT
    (   5)
    3609.34
      81
      |
    6996.81
    XISCAN
    (   6)
    3017.54
      81
      |
    210000
XMLIN: DB2XML
   TX_IDX
```

Performance suggestions:

- Over time, database updates might cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or reorganizing the index.
- When two or more tables are being accessed, access to the inner table via an index can be made more efficient by providing an index on the join column of the outer table.
- If statistics are not current, update them using the RUNSTATS command.

XSCAN operator

This operator is used to navigate XML fragments to evaluate XPath expressions and to extract document fragments if needed.

Operator name: XSCAN

Represents: This operator processes node references passed by a nested-loop join operator (NLJOIN). It is not represented with a direct input in the access plan.

For example, consider the following XQuery statement:

```
XQUERY for $i in db2-fn:xmlcolumn("MOVIES.XMLCOL")//actor return $i
```

The access plan for this statement as provided by the db2exfmt utility shows an XSCAN operator processing document node references retrieved by a table scan on the table TELIAZ.MOVIES. The XSCAN operator in this case returns actor node references found within the TELIAZ.MOVIES.XMLCOL collection.

```
         Rows
        RETURN
        (   1)
         Cost
          I/O
          |
          180
        NLJOIN
        (   2)
        10137.9
         1261
         /-+\
      180      1
    TBSCAN  XSCAN
    (   3)  (   4)
    21.931   56.2
       1       7
      |
      180
TABLE: TELIAZ
    MOVIES
```

Chapter 6. Visual Explain concepts

This section contains Visual Explain conceptual information.

Access plan

Certain data is necessary to resolve an *explainable statement*. An *access plan* specifies an order of operations for accessing this data.

An access plan lets you view statistics for selected tables, indexes, or columns; properties for operators; global information such as table space and function statistics; and configuration parameters relevant to optimization. With Visual Explain, you can view the access plan for an SQL or XQuery statement in graphical form.

The optimizer produces an access plan whenever you compile an explainable SQL or XQuery statement. This happens at prep/bind time for static statements, and at run time for dynamic statements.

It is important to understand that an access plan is an *estimate* based on the information that is available. The optimizer bases its estimations on information such as the following:

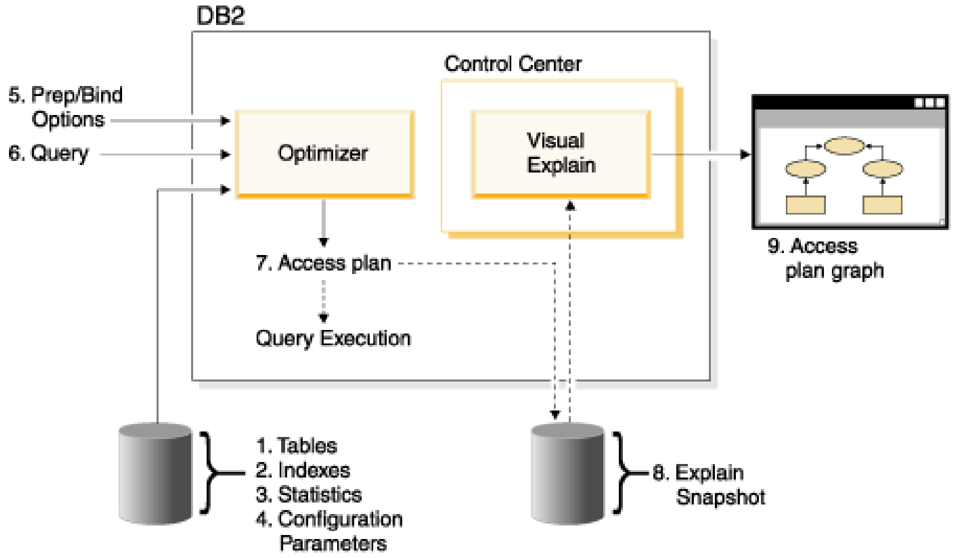
- Statistics in system catalog tables (if statistics are not current, update them using the RUNSTATS command.)
- Configuration parameters
- Bind options
- The query optimization class

Cost information associated with an access plan is the optimizer's *best estimate* of the resource usage for a query. The actual elapsed time for a query might vary depending on factors outside the scope of the database manager (for example, the number of other applications running at the same time). Actual elapsed time can be measured while running the query, by using performance monitoring.

Access plan graph

Visual Explain uses information from a number of sources in order to produce an access plan graph

Based on various inputs, as shown in the illustration below, the optimizer chooses an access plan, and Visual Explain displays it in an *access plan graph*. The nodes in the graph represent tables and indexes and each operation on them. The links between the nodes represent the flow of data.



Access plan graph node

The access plan graph consists of a tree displaying *nodes*.

These nodes represent:
- Tables, shown as rectangles
- Indexes, shown as diamonds
- Operators, shown as octagons (8 sides). TQ operators, shown as parallelograms
- Table functions, shown as hexagons(6 sides).

Clustering

Over time, updates might cause rows on data pages to change location lowering the degree of *clustering* that exists between an index and the data pages.

Reorganizing a table with respect to a chosen index re-clusters the data. A clustered index is most useful for columns that have range predicates because it allows better sequential access of data in the base table. This results in fewer page fetches, since like values are on the same data page.

In general, only one of the indexes in a table can have a high degree of clustering.

To check the degree of clustering for an index, double-click on its node to display the Index Statistics window. The cluster ratio or cluster factor values are shown in this window. If the value is low, consider reorganizing the table's data.

Container

A *container* is a physical storage location of the data.

It is associated with a table space, and can be a file or a directory or a device. Containers are numbered sequentially, starting at 0.

Cost

Cost, in the context of access plans access plans, is the estimated total resource usage necessary to execute the access plan for a statement (or the elements of a statement).

Cost is derived from a combination of CPU cost (in number of instructions) and I/O (in numbers of seeks and page transfers).

The unit of cost is the *timeron*. A timeron does not directly equate to any actual elapsed time, but gives a rough relative estimate of the resources (cost) required by the database manager to execute two plans for the same query.

The cost shown in each operator node of an access plan graph is the cumulative cost, from the start of access plan execution up to and including the execution of that particular operator. It does not reflect factors such as the workload on the system or the cost of returning rows of data to the user.

Cursor blocking

Cursor blocking is a technique that reduces overhead by having the database manager retrieve a *block* of rows in a single operation.

These rows are stored in a cache in the DB2 client while they are processed. The cache is allocated when an application issues an OPEN CURSOR request, and is de-allocated when the cursor is closed. When all the rows have been processed, another block of rows is retrieved.

Use the BLOCKING option on the PREP or BIND commands along with the following parameters to specify the type of cursor blocking:

UNAMBIG
For cursors that are specified with the FOR READ ONLY clause, blocking occurs.

Cursors that are not declared with the FOR READ ONLY or FOR UPDATE clause which are not *ambiguous* and are *read-only* will be blocked. *Ambiguous* cursors will not be blocked.

ALL For cursors that are specified with the FOR READ ONLY clause or are not specified as FOR UPDATE, blocking occurs.

NO Blocking does not occur for any cursor.

For the definition of a read-only cursor and an ambiguous cursor, refer to the DECLARE CURSOR statement.

Database-managed table space

There are two types of *table spaces* that can exist in a database: *database-managed space (DMS)*, and *system-managed space (SMS)*.

DMS table spaces are managed by the database manager. and are designed and tuned to meet its requirements.

The DMS table space definition includes a list of files (or devices) into which the database data is stored in its DMS table space format.

You can add pre-allocated files (or devices) to an existing DMS table space in order to increase its storage capacity. The database manager automatically rebalances existing data in all the *containers* belonging to that table space.

DMS and SMS table spaces can coexist in the same database.

Dynamic SQL or XQuery

Dynamic SQL or XQuery statements are statements that are prepared and executed within an application program while the program is running.

In *dynamic SQL or XQuery*, either:

- You issue the SQL or XQuery statement interactively, using CLI or CLP
- The SQL or XQuery source is contained in host language variables that are embedded in an application program.

When the database manager runs a dynamic SQL or XQuery statement, it creates an *access plan* that is based on current catalog statistics and configuration parameters. This access plan might change from one execution of the statements application program to the next.

The alternative to dynamic SQL or XQuery is *static SQL or XQuery*.

Explain snapshot

With Visual Explain, you can examine the contents of an explain snapshot. An *explain snapshot* is compressed information that is collected when an SQL statement is explained.

It is stored as a binary large object (BLOB) in the EXPLAIN_STATEMENT table, and contains the following information:

- The internal representation of the access plan, including its operators and the tables and indexes accessed
- The decision criteria used by the optimizer, including statistics for database objects and the cumulative cost for each operation.

An explain snapshot is required if you want to display the graphical representation of an SQL statement's access plan. To ensure that an explain snapshot is created:

1. Explain tables must exist in the database manager to store the explain snapshots. For information on how to create these tables, see Creating explain tables in the online help.
2. For a package containing static SQL or XQuery statements, set the EXPLSNAP option to ALL or YES when you bind or prep the package. You will get an explain snapshot for each explainable SQL statement in the package. For more information, see the BIND and PREP commands.
3. For dynamic SQL statements, set the EXPLSNAP option to ALL when you bind the application that issues them, or set the CURRENT EXPLAIN SNAPSHOT special register to YES or EXPLAIN before you issue them interactively. For more information, see the CURRENT EXPLAIN SNAPSHOT special register and the SET CURRENT EXPLAIN SNAPSHOT statement.

Explainable statement

An *explainable statement* is an SQL or XQuery statement for which an explain operation can be performed.

Explainable SQL or XQuery statements are:

- DELETE
- INSERT
- MERGE
- REFRESH TABLE
- SELECT
- SET INTEGRITY
- UPDATE
- VALUES

Explained statement

An *explained statement* is an SQL or XQuery statement for which an explain operation has been performed.

Explained statements are shown in the Explained Statements History window.

Operand

An operand is an entity on which an operation is performed.

For example, a table or an index is an operand of various operators such as TBSCAN and IXSCAN.

Operator

An *operator* is either an action that must be performed on data, or the output from a table or an index, when the access plan for an SQL or XQuery statement is executed.

The following operators can appear in the access plan graph:

DELETE
Deletes rows from a table.

EISCAN
Scans a user defined index to produce a reduced stream of rows.

FETCH
Fetches columns from a table using a specific record identifier.

FILTER
Filters data by applying one or more predicates to it.

GENROW
Generates a table of rows.

GRPBY
Groups rows by common values of designated columns or functions, and evaluates set functions.

HSJOIN
Represents a hash join, where two or more tables are hashed on the join columns.

INSERT
Inserts rows into a table.

IXAND
ANDs together the row identifiers (RIDs) from two or more index scans.

IXSCAN
Scans an index of a table with optional start/stop conditions, producing an ordered stream of rows.

MSJOIN
Represents a merge join, where both outer and inner tables must be in join-predicate order.

NLJOIN
Represents a nested loop join that accesses an inner table once for each row of the outer table.

RETURN
Represents the return of data from the query to the user.

RIDSCN
Scans a list of row identifiers (RIDs) obtained from one or more indexes.

RPD (Remote PushDown)
An operator for remote plans. It is very similar to the SHIP operator in Version 8 (RQUERY operator in previous versions), except that it does not contain an SQL or XQuery statement.

SHIP Retrieves data from a remote database source. Used in the federated system.

SORT Sorts rows in the order of specified columns, and optionally eliminates duplicate entries.

TBSCAN
Retrieves rows by reading all required data directly from the data pages.

TEMP Stores data in a temporary table to be read back out (possibly multiple times).

TQ Transfers table data between database agents.

UNION
Concatenates streams of rows from multiple tables.

UNIQUE
Eliminates rows with duplicate values, for specified columns.

UPDATE
Updates rows in a table.

XISCAN
Scans an index of an XML table.

XSCAN
Navigates an XML document node subtrees.

XANDOR
Allows ANDed and ORed predicates to be applied to multiple XML indexes.

Optimizer

The *optimizer* is the component of the SQL compiler that chooses an access plan for a data manipulation language (DML) SQL statement.

It does this by modeling the execution cost of many alternative access plans, and choosing the one with the minimal estimated cost.

Package

A *package* is an object stored in the database that includes the information needed to process the SQL statements associated with one source file of an application program.

It is generated by either:

- Precompiling a source file with the PREP command
- Binding a bind file that was generated by the precompiler with the BIND command.

Predicate

A *predicate* is an element of a search condition that expresses or implies a comparison operation. Predicates are included in clauses beginning with WHERE or HAVING.

For example, in the following SQL statement:

```
SELECT * FROM SAMPLE
  WHERE NAME = 'SMITH' AND
  DEPT = 895 AND YEARS > 5
```

The following are predicates: NAME = ’SMITH’; DEPT = 895; and YEARS > 5.

Predicates fall into one of the following categories, ordered from most efficient to least efficient:

1. Starting and stopping conditions bracket (narrow down) an index scan. (These conditions are also called range-delimiting predicates.)
2. Index-page (also known as index sargable) predicates can be evaluated from an index because the columns involved in the predicate are part of the index key.
3. Data-page (also known as data sargable) predicates cannot be evaluated from an index, but can be evaluated while rows remain in the buffer.
4. Residual predicates typically require I/O beyond the simple accessing of a base table, and must be applied after data is copied out of the buffer page. They include predicates that contain subqueries, or those that read LONG VARCHAR or LOB data stored in files separate from the table.

When designing predicates, you should aim for the highest *selectivity* possible so that the fewest rows are returned.

The following types of predicates are the most effective and the most commonly used:

- A *simple equality join predicate* is required for a merge join. It is of the form table1.column = table2.column, and allows columns in two different tables to be equated so that the tables can be joined.
- A *local predicate* is applied to one table only.

Query optimization class

A *query optimization class* is a set of query rewrite rules and optimization techniques for compiling queries.

The primary query optimization classes are:

1 Restricted optimization. Useful when memory and processing resources are severely restrained. Roughly equivalent to the optimization provided by Version 1.

2 Slight optimization. Specifies a level of optimization higher than that of Version 1, but at significantly less optimization cost than levels 3 and above, especially for very complex queries.

3 Moderate optimization. Comes closest to matching the query optimization characteristics of DB2 for z/OS®.

5 Normal optimization. Recommended for a mixed environment using both simple transactions and complex queries.

7 Normal optimization. The same as query optimization 5 except that it does not reduce the amount of query optimization for complex dynamic queries.

Other query optimization classes, to be used only under special circumstances, are:

0 Minimal optimization. Use only when little or no optimization is required (that is, for very simple queries on well-indexed tables).

9 Maximum optimization. Uses substantial memory and processing resources. Use only if class 5 is insufficient (that is, for very complex and long-running queries that do not perform well at class 5).

In general, use a higher optimization class for static queries and for queries that you anticipate will take a long time to execute, and a lower optimization class for simple queries that are submitted dynamically or that are run only a few times.

To set the query optimization for dynamic SQL or XQuery statements, enter the following command in the command line processor:

```
SET CURRENT QUERY OPTIMIZATION = n;
```

where 'n' is the desired query optimization class.

To set the query optimization for static SQL or XQuery statements, use the QUERYOPT option on the BIND or PREP commands.

Sample XML document fragment for Explain XML operators

This sample XML document fragment is used in the discussion of the Explain XML operators XSCAN, XISCAN, and XANDOR.

```
<emp id='12345' salary='60000'>
 <name>
  <first>William</first>
  <last>Murphy</last>
 </name>
 <spouse>
  <name>
   <first>Cecilia</first>
   <last>Murphy</last>
  </name>
 </spouse>
 <dept id='K55'>
  Finance
 </dept>
</emp>
<emp id='12345' salary='40000'>
 <name>
  <first>Patricio</first>
  <last>Murphy</last>
 </name>
 <dept id='A15'>
  Sales
 </dept>
</emp>
<emp id='12346' salary='70000'>
 <name>
  <first>Victoria</first>
  <last>Zubiri</last>
 </name>
 <dept id='B11'>
  Marketing
 </dept>
</emp>
```

Selectivity of predicates

Selectivity refers to the probability that any row will satisfy a predicate (that is, be true).

For example, a selectivity of 0.01 (1%) for a predicate operating on a table with 1,000,000 rows means that the predicate returns an estimated 10,000 rows (1% of 1,000,000), and discards an estimated 990,000 rows.

A highly selective predicate (one with a selectivity of 0.10 or less) is desirable. Such predicates return fewer rows for future operators to work on, thereby requiring less CPU and I/O to satisfy the query.

Example: Suppose that you have a table of 1,000,000 rows, and that the original query contains an 'ORDER BY' clause requiring an additional sorting step. With a predicate that has a selectivity of 0.01, the sort would have to be done on an estimated 10,000 rows. However, with a less selective predicate of 0.50, the sort would have to be done on an estimated 500,000 rows, thus requiring more CPU and I/O time.

Star join

A set of joins are considered to be a star join when a fact table (large central table) is joined to two or more dimension tables (smaller tables containing descriptions of the column values in the fact table).

A Star join is comprised of 3 main parts:

- Semijoins
- Index ANDing of the results of the Semijoins
- Completing the semijoins.

It shows up as two or more joins feeding an "IXAND operator" on page 52 operator.

A Semijoin is a special form of join in which the result of the join is only the Row Identifier (RID) of the inner table, instead of the joining of the inner and outer table columns.

Star joins use Semijoins to supply Row Identifiers to an Index ANDing operator. The Index ANDing operator accumulates the filtering affect of the various joins. The output from the Index ANDing operator is fed into an Index ORing operator, which orders the Row Identifiers, and eliminates any duplicate rows that may have resulted from the joins feeding the Index ANDing operator. The rows from the fact table are then fetched, using a Fetch operator. Finally, the reduced fact table is joined to all of the dimension tables, to complete the joins.

Performance suggestions:

- Create indexes on the fact table for each of the dimension table joins.
- Ensure the sort heap threshold is high enough to allow allocating the Index ANDing operator’s bit filter. For star joins, this could require as much as 12MB, or 3000 4K pages. For Intra-partition parallelism, the bit filter is allocated from the same shared memory segment as the shared sort heap, and it is bounded by the *sortheap* database configuration parameter and the *sheapthres_shr* database configuration parameter.
- Apply filtering predicates against the dimension tables. If statistics are not current, update them using the runstats command.

Static SQL or XQuery

A *static SQL or XQuery* statement is embedded within an application program. All these embedded statements must be precompiled and bound into a *package* before the application can be executed.

To execute XQuery expressions in static SQL, use the XMLQUERY function.

When the database manager compiles these statements, it creates an access plan for each one that is based on the catalog statistics and configuration parameters at the time that the statements were precompiled and bound.

These access plans are always used when the application is run; they do not change until the package is bound again.

The alternative to static SQL or XQuery is *dynamic SQL or XQuery*.

System-managed table spaces

There are two types of *table spaces* that can exist in a database: *system-managed space (SMS)* and *database-managed space (DMS).*

An SMS table space is managed by the operating system, which stores the database data into a space that is assigned when a table space is created. The table space definition includes a list of one or more of the directory paths where this data is stored.

The file system manages the allocation and management of media storage.

SMS and DMS table spaces can coexist in the same database.

Table spaces

It is easier to manage very large databases if you partition them into separately managed parts called *table spaces*. A table space lets you assign the location of data to particular logical devices or portions thereof.

For example, when creating a table you can specify that its indexes or its long columns with long or large object (LOB) data be kept away from the rest of the table data.

A table space can be spread over one or more physical storage devices (containers) for increased performance. However, it is recommended that all the devices or containers within a table space have similar performance characteristics.

A table space can be managed in two different ways: as a *system-managed space (SMS)* or as a *database-managed space (DMS)*.

Visual Explain

Visual Explain lets you view the access plan for explained SQL or XQuery statements as a graph. You can use the information available from the graph to tune your queries for better performance.

Important: Access to Visual Explain through the Control Center tools has been deprecated in Version 9.7 and might be removed in a future release. For more information, see the "Control Center tools and DB2 administration server (DAS) have been deprecated" topic in the *What's New for DB2 Version 9.7* book. Accessing Visual Explain functionality through the IBM Data Studio toolset has not been deprecated.

You can use Visual Explain to:

- View the statistics that were used at the time of optimization. You can then compare these statistics to the current catalog statistics to help you determine whether rebinding the package might improve performance.
- Determine whether or not an index was used to access a table. If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- View the effects of performing various tuning techniques by comparing the before and after versions of the access plan graph for a query.
- Obtain information about each operation in the access plan, including the total estimated cost and number of rows retrieved (cardinality).

An *access plan* graph shows details of:

- Tables (and their associated columns) and indexes
- Operators (such as table scans, sorts, and joins)
- Table spaces and functions.

Note: Note that Visual Explain cannot be invoked from the command line, but only from various database objects in the Control Center.

To start Visual Explain:

- From the Control Center, right-click a database name and select either **Show Explained Statements History** or **Explain Query**.

- From the Command Editor, execute an explainable statement on the Interactive page or the Script page.
- From the Query Patroller, click **Show Access Plan** from either the Managed Queries Properties notebook or from the Historical Queries Properties notebook.

Chapter 7. Visual Explain tasks

This section contains a list of related tasks that you can also perform using Visual Explain.

Creating an access plan using the Command Editor

Use the Command Editor to generate, edit, execute, and manipulate SQL and XQuery statements, IMS™ commands, and DB2 commands.

You can also use the Command Editor to work with the resulting output and to view a graphical representation of the access plan for explained SQL statements. You can execute commands and SQL statements on DB2 databases for Linux® and Windows®, for z/OS and OS/390® systems and subsystems, and for IMSplexes.

To create an access plan using the Command Editor:

1. Open the Command Editor: To open a stand-alone Command Editor, select **Start –> Programs –> IBM DB2 –> Command Line Tools –> Command Editor**.
2. Select either the Interactive or Script tab, and do the following:
   a. Connect to a database. (Type the connect command in the text area and select **Execute** from the Interactive or Script menu, depending on which page you selected in step 2., or click on the ⚙ icon, or press the **Ctrl+Enter** keys to execute the command.)
   b. To create an access plan without executing the statement, type an explainable statement explainable statement in the text area and select **Create access plan**, from the Interactive or Script menu, or click on the ⁂ icon. The access plan graph is displayed on the Access Plan page.

      You can also select an explainable statement from an existing script.
3. To create an access plan and also execute the statement:
   a. Select **Options** from the Interactive or Script menu. The Command Center Options notebook opens. Click on the Access Plan tab. Select the **Automatically generate access plan** check box.
   b. Type an *explainable statement* in the text area or select an existing statement.

      Select **Execute** from the Interactive or Script menu, or click the ⚙ icon. The results are displayed on the Results page. To view the generated access plan, click on the Access Plan tab.

Viewing a graphical representation of an access plan

Use the Access Plan Graph window to view a graphical representation of the *access plan* of an explained SQL or XQuery statement.

The nodes in the graph represent tables and indexes and each operation on them. The links between the nodes represent the flow of data.

Tasks

- Use the Statement menu to print the graph, to dynamically explain an SQL or XQuery statement, to view the text or optimized text, or to view optimization parameters or statistics.
- Use the Node menu to view details or statistics on the nodes, or to get additional help on each of the operators.
- Use the View menu to change the graph settings or to see an overview of the graph. This is particularly useful for large graphs.

From this window, you can view details about the following objects:

- Table spaces and table space statistics
- Functions and function statistics
- Operators
- Partitioned databases
- Operands
  - Column distribution statistics
  - Index and index statistics
  - Page fetch pairs statistics
  - Column groups
  - Referenced columns, referenced column groups, and referenced column statistics
  - Table function statistics and table statistics

To open the Access Plan Graph window, use one of the following methods:

1. Open either the Explainable Statements, or the Explained Statements History window. Select **Statement–>Show Access Plan**. The Access Plan Graph window opens.
2. Invoke **Explain Query** from either the Explainable Statements or the Explained Statements History window. The Explain Query statement window opens as a result of the dynamic explain.

Reading the contents of the Access Plan Graph window

Top area of the window

The top area of the Access Plan Graph window identifies the statement whose access plan is displayed on the graph.

This part of the window also shows:

- The statement's explain date, time, package name, and version.
- If the Federated function was enabled at the time the statement was created.
- Its total estimated cost.
- The type of parallelism of the system in which this statement is explained. It can be one of the following types:
  - None
  - Intra-partition parallelism
  - Inter-partition parallelism
  - Full parallelism (intra-partition and inter-partition)

The graph

The nodes in the graph represent operands (tables, indexes, or table functions), and the operators that act on them. To view detailed statistical information for a node, double-click on it.

To view the information shown in the graph in more detail, drag the zoom slider up or down.

Float values might be presented in scientific notation.

Troubleshooting Tips

- Retrieving the access plan when using LONGDATACOMPAT
- Visual Explain support for earlier and later releases

Viewing the history of previously explained query statements

Use the Explained Statements History window to view the history of previously explained SQL or XQuery statements for a selected database.

Each entry is an explained statement that is associated with either:

- A static SQL or XQuery statement in a package
- A dynamic SQL or XQuery statement.

Tasks

- Use the Statement menu to view a graphical representation of an access plan, to dynamically explain a query statement, to view text for a query statement, or to change or remove a query statement.
- Use the View menu, or the icons on the secondary toolbar to sort, filter, or customize the explainable statements. You can also save the contents of this window using the options in this menu.

To open Explained Statements History window, do one of the following:

- From the Control Center, expand the object tree until you find the Databases folder, expand the folder until you find the database you want, and then do one of the following:
  - Right-click the database and select **Show Explained Statements History** from the pop-up menu, or select **Selected–>Show Explained Statements History**.
  - Highlight the database and select **Selected–>Show Explained Statements History**.
- From the Control Center, expand the object tree until you find the Packages folder (under the Application Objects folder). Then:
  - click the Packages folder. Any existing package objects are displayed on the right side of the window.
  - Right-click the package that you want, and select **Show Explained Statements History** from the pop-up menu; or highlight the package and select **Selected–>Show Explained Statements History**; or simply double-click the package.
- From the Explainable Statements window, select **Statement–>Show Explained Statements History**.

  If a statement is selected in the Explainable Statements window, the Explained Statements History window shows all of the explained statements that are related to the selected SQL statements.

If no statement is selected, the Explained Statements History window shows all the explained statements that are related to the package that the explainable statements are in.

The Explained Statements History window might or might not contain explained statements, depending on whether the explain tables exist.

Reading the contents of the Explained Statements History window

The columns in the window provide the following information about the query statements that have been explained:

Package name
The name of the package that either:

- Contains the SQL or XQuery statement (in the case of a static query)
- Issued the SQL or XQuery statement (in the case of a dynamic query).

Package creator
The user ID of the user who created the package.

Package version
The version number of the package.

Explain snapshot
States whether an explain snapshot has been taken for the SQL or XQuery statement. (If it has not, you cannot view an access plan graph for the statement.)

Latest bind
If the statement is contained in a package, this field indicates whether or not the statement is associated with the latest bound package.

Dynamic explain
States whether the explained query statement was dynamic. (If it was not, it was a static or SQL or XQuery statement in a package.)

Explain date
The date when the statement had an explain operation performed on it.

Explain time
The time when the statement had an explain operation performed on it.

Total cost
The estimated total cost (in timerons) of the statement.

Statement number
The line number of the SQL or XQuery statement in the source module of the application program.

Section number
The number of the section within the package that is associated with the SQL or XQuery statement.

Query number
The query number that is associated with the statement.

Query tag
The query tag that is associated with the statement.

Query text
The first 100 characters of the original SQL or XQuery statement. (Use the

scroll bar at the bottom of the window to scroll through it.) To view the complete SQL or XQuery statement, select Statement–>Show Query Text.

Remarks
Any remarks associated with the statement. (For example, for a static query statement, the remark associated with the package containing the statement.)

Troubleshooting Tips

- "Retrieving the access plan when using LONGDATACOMPAT" on page 81
- "Visual Explain support for earlier and later releases" on page 82

Viewing explainable statements for a package

Use the Explainable Statements window to view the explainable query statements for a selected *package*.

If an explain snapshot has been taken for a statement, you can use this list to view additional information about that statement (such as its total cost and a graphical view of its access plan).

Tasks

- Use the Statement menu to view the history of previously explained SQL or XQuery statements, to view a graphical representation of the access plan, to dynamically explain a query statement, and to view text for a query statement.
- Use the View menu, or the icons on the secondary toolbar to sort, filter, or customize the explainable statements. You can also save the contents of this window using the options in this menu.

To open the Explainable Statements window, do the following:

- From the Control Center, expand the object tree until you find the Packages folder (under the Application Objects folder).
- click the Packages folder. Any existing package objects are displayed in the pane on the right side of the window.
- Do one of the following:
  - Right-click the package you want and select **Show Explainable Statements** from the pop-up menu.
  - Highlight the package and select **Selected–>Show Explainable Statements**.
  - Double-click the package.

Reading the contents of the Explainable Statements window

The columns in the window provide the following information about SQL or XQuery statements:

Statement number
The line number of the SQL or XQuery statement in the source module of the application program. For static queries, this number corresponds to the STMTNO column in the SYSCAT.STATEMENTS table.

Section number
The number of the section within the package that is associated with the SQL or XQuery statement.

Explain snapshot
States whether an explain snapshot has been taken for the SQL or XQuery statement. (If it has not been taken, you cannot view an access plan graph for the statement.)

Total cost
The estimated total cost(in timerons) of returning the query results for the selected SQL or XQuery statement. (Available only if the packagecontaining the statement has been explained previously.)

Query text
The first 100 characters of the query statement. (Use the scroll bar at the bottom of the window to scroll through it.) To view the complete SQL or XQuery statement, select **Statement–>Show Query Text**.

Troubleshooting Tips

- "Retrieving the access plan when using LONGDATACOMPAT" on page 81
- "Visual Explain support for earlier and later releases" on page 82

Guidelines for creating indexes

Creating appropriate indexes allows the optimizer to choose an index scan for those cases where it would be more efficient than a table scan.

Some guidelines for creating indexes include:

- Define primary keys and unique indexes wherever they apply.
- Create an index on any column that the query uses to join tables (*join predicate*).
- Create an index on any column from which you search for particular values on a regular basis.
- Create an index on columns that are commonly used in ORDER BY clauses.
- Ensure that you have used predicates that retrieve only the data you need. For example, ensure that the *selectivity of predicates* value for the predicates represents the portion of the table that you want returned.
- When creating a multicolumn index, the first columns of the index should be the ones that are used most often by the predicates in your query.
- Ensure that the disk and update maintenance overhead an index introduces will not be too high.

Out-of-date access plans

If your access plans are out-of-date, you need to update the statistics; then rebind the package.

Symptom
The STATS_TIME row indicates that the statistics are not updated.

Possible cause
The optimizer used default values. (These default values are displayed with the keyword ″default″.) This situation can result in an out-of-date access plan.

Action
It is recommended that you use the runstats command to update the statistics; then rebind the package.

Retrieving the access plan when using LONGDATACOMPAT

If you cannot retrieve the access plan when using LONGDATACOMPAT, create a database alias and try again.

Symptom

No explained statement history or access plan can be displayed using Visual Explain.

Possible cause

If the value for LONGDATACOMPAT is set to 1 in the db2cli.ini file, the Visual Explain access plan can be generated but cannot be retrieved.

Action

As a work around, a database alias can be created for that database with LONGDATACOMPAT set to 0. For example:

```
DB2 UPDATE CLI CFG FOR SECTION db-alias-name USING LONGDATACOMPAT 0
```

To check the CLI configuration values, the following command can be used:

```
GET CLI CONFIGURATION [AT GLOBAL LEVEL] [FOR SECTION section-name]
```

For instance, if the database name is called sample:

```
GET CLI CONFIGURATION FOR SECTION sample
```

Using RUNSTATS

The optimizer uses the catalog tables from a database to obtain information about the database, the amount of data in it, and other characteristics, and uses this information to choose the best way to access the data.

If current statistics are not available, the optimizer might choose an inefficient access plan based on inaccurate default statistics.

It is highly recommended that you use the runstats command to collect current statistics on tables and indexes, especially if significant update activity has occurred or new indexes have been created since the last time the runstats command was executed. This provides the optimizer with the most accurate information with which to determine the best access plan.

Be sure to use runstats *after* making your table updates; otherwise, the table might appear to the optimizer to be empty. This problem is evident if cardinality on the Operator Details window equals zero. In this case, complete your table updates, rerun the runstats command and recreate the explain snapshots for affected tables.

Note:

- Use runstats on all tables and indexes that might be accessed by a query.
- The quantile and frequent value statistics determine when data is unevenly distributed. To update these values, use runstats on a table with the WITH DISTRIBUTION clause.
- In addition to statistics, other factors (such as the ordering of qualifying rows, table size, and buffer pool size) might influence how an access plan is selected.

- Applications should be rebound (and their statements optionally re-explained) after you run the runstats command or change configuration parameters.

The runstats command (which can be entered from the CLP prompt) can provide different levels of statistics as shown in the following syntax:

Basic Statistics

Table:

```
RUNSTATS ON TABLE tablename
```

Index:

```
RUNSTATS ON TABLE tablename FOR INDEXES ALL
```

Both tables and indexes:

```
RUNSTATS ON TABLE tablename AND INDEXES ALL
```

Enhanced Statistics

Table:

```
RUNSTATS ON TABLE tablename WITH DISTRIBUTION
```

Index:

```
RUNSTATS ON TABLE tablename FOR DETAILED INDEXES ALL
```

Both tables and indexes:

```
RUNSTATS ON TABLE tablename WITH DISTRIBUTION AND
DETAILED INDEXES ALL
```

Note: In each of the above commands, the tablename *must* be fully qualified with the schema name.

Visual Explain support for earlier and later releases

The *snapshots* generated by Version 9 are different from those generated by Version 8.

If you are running Visual Explain on a Version 9 client accessing a Version 8 database, Visual Explain does handle the Version 8 snapshots. Visual Explain supports earlier release compatibility.

However, if you are running Visual Explain on a Version 8 client accessing a Version 9 database, Visual Explain returns an error when it tries to parse the Version 9 data. Visual Explain does not support this upward level compatibility since the snapshots generated by Version 9 are different from those generated by Version 8.

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 1. 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 1. 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 2. 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 2. 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 3. 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 85.

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 sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided ″AS IS″, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java™ and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX® is a registered trademark of The Open Group in the United States and other countries.
- Intel®, Intel logo, Intel Inside®, Intel Inside logo, Intel® Centrino®, Intel Centrino logo, Celeron®, Intel® Xeon®, Intel SpeedStep®, Itanium®, and Pentium® are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft®, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

Printed in USA

SC27-2462-00

Spine information:

IBM DB2 9.7 for Linux, UNIX, and Windows

Visual Explain Tutorial