DB2 9 DBA certification exam 731 prep, Part 4: Monitoring DB2 activity

Skill Level: Introductory

Roger E. Sanders (rsanders@netapp.com)
Senior Manager - IBM Alliance Engineering
Network Appliance, Inc.

05 Jul 2006

This tutorial introduces you to the set of monitoring tools that are available with DB2® 9 and to show you how each are used to monitor how well (or how poorly) your database system is operating. This is the fourth tutorial in a series of seven that you can use to help prepare for the DB2 9 for Linux®, UNIX®, and Windows™ Database Administration Certification (Exam 731).

Section 1. Before you start

About this series

If you are preparing to take the DB2 DBA certification exam 731, you've come to the right place -- a study hall, of sorts. This series of seven DB2 certification preparation tutorials covers the major concepts you'll need to know for the test. Do your homework here and ease the stress on test day.

About this tutorial

Tuning and configuring a DB2 database can be a complex process that sometimes overwhelms new DBAs. There are, however, a great number of tools, functions, and applications included with DB2 that, once mastered, make this task simple.

This tutorial is designed to introduce you to the set of monitoring tools that are available with DB2 9 and to show you how each are used to monitor how well (or how poorly) your database system is operating. In this tutorial, you will learn:

- How the database system monitor works
- How snapshot information is collected
- How event monitors are created and how event monitor data is collected
- How the health monitor and the Health Center are used
- How comprehensive explain data and explain snapshot data differ
- How comprehensive explain data and explain snapshot data is collected
- How Visual Explain is used to view explain snapshot data

This is the fourth tutorial in a series of seven tutorials to help you prepare for the DB2 9 for Linux, UNIX, and Windows Database Administration Certification (Exam 731). The material in this tutorial primarily covers the objectives in Section 4 of the exam, entitled "Analyzing DB2 Activity." You can view these objectives at: http://www-03.ibm.com/certify/tests/obj731.shtml.

Objectives

After completing this tutorial, you should be able to:

- Capture snapshots using `GET SNAPSHOT` or SQL functions
- Create and activate event monitors
- Configure the health monitor using the Health Center
- Capture and analyze explain/Visual Explain information
- Identify the functions of DB2's problem determination tools (for example, db2pd and db2mtrk)

Prerequisites

To take the DB2 9 DBA exam, you must have already passed the DB2 9 Fundamentals exam 730. We recommend that you take the DB2 Fundamentals tutorial series before starting this series.

To help you understand some of the material presented in this tutorial, you should be familiar with the following terms:

- **Structured Query Language (SQL):** A standardized language used to define objects and manipulate data in a relational database.
- **DB2 optimizer:** A component of the SQL precompiler that chooses an access plan for a Data Manipulation Language (DML) SQL statement by modeling the execution cost of several alternative access plans and choosing the one with the minimal estimated cost.

System requirements

You do not need a copy of DB2 to complete this tutorial. However, you will get more out of the tutorial if you download the free trial version of IBM DB2 9 to work along with this tutorial.

Section 2. Capturing snapshot data

The database system monitor

Database monitoring is a vital activity that, when performed on a regular basis, provides continuous feedback on the health of a database system. Because database monitoring is such an integral part of database administration, DB2 9 comes equipped with a monitoring utility known as the database system monitor. Although the name "database system monitor" suggests that only one monitoring tool is provided, in reality the database system monitor is composed of two distinct tools that can be used to capture and return system monitor information: a *snapshot monitor* and one or more *event monitors*. The snapshot monitor allows you to capture a picture of the state of a database at a specific point in time while event monitors capture and log data as specific database events occur. Information collected by both tools is stored in entities that are referred to as monitor elements (or data elements). Each monitor element used is identified by a unique name and is designed to hold a specific type of information. The following types of elements are used to store monitor data:

- **Counters.** Counters keep a total count of the number of times an activity or event has occurred. Counter values increase throughout the life of the monitor; often a counter monitor element is resettable. An example of a counter element would be the total number of SQL statements that have been executed against a database.

- **Gauges.** Gauges keep a count of the number times an activity or event has occurred at a specific point in time. Unlike counter values, gauge values can go up or down, and their value at any given point in time is usually dependent upon the level of database activity. An example of a gauge element would be the number of applications that are currently connected to a database.

- **Watermarks.** Watermarks indicate the highest (maximum) or lowest (minimum) value an item has seen since monitoring began. An example of a watermark element would be the largest number of rows that were affected by an update operation.

- **Information.** As the name implies, information elements provide reference-type details of all monitoring activities performed. Examples of information elements would include buffer pool names, database names and aliases, path details, etc.

- **Timestamps.** Timestamps indicates the date and time an activity or event took place. Timestamp values are provided as the number of seconds and microseconds that have elapsed since January 1, 1970. An example of a timestamp element would be the date and time the first connection to a database was established.

- **Time.** Time elements keep track of the amount of time that was spent performing an activity or event. Time values are provided as the number of seconds and microseconds that have elapsed since the activity or event was started and some time elements are resettable. An example of a time element would be the amount of time that was spent performing a sort operation.

The database system monitor can utilize any combination of these elements to capture monitor data, and once collected, several methods can be used to present the data stored in each element used; for both snapshot monitors and event monitors, you have the option of storing all data collected in files or database tables, viewing it on-screen, or processing it using a custom application. (The database system monitor returns monitor data to a client application using a self-describing data stream. With a snapshot monitoring application you call the appropriate snapshot APIs to capture a snapshot and then process the data stream returned; with an event monitoring application, you prepare to receive the data produced via a file or a named pipe, activate the appropriate event monitor, and process the data stream as it is received.)

The snapshot monitor

The snapshot monitor is designed to collect information about the state of a DB2 UDB instance and the databases it controls *at a specific point in time* (in other words, at the time the snapshot is taken). Snapshots are useful for determining the status of a database system, and when taken at regular intervals, they can provide valuable information that can be used to observe trends and identify potential problem areas. Snapshots can be taken by executing the `GET SNAPSHOT` command from the DB2 Command Line Processor (CLP), by using the appropriate snapshot table functions in a query, or by using the snapshot monitor APIs in a C or C++ application. Additionally, snapshots can be tailored to return specific types of monitoring data values (for example, a snapshot could be configured to return just information about buffer pools).

Snapshot monitor switches

Often, the collection of system monitor data requires additional processing overhead. For example, in order to calculate the execution time of SQL statements, the DB2 Database Manager must make a call to the operating system to obtain timestamps before and after any SQL statement is executed. These types of system calls can be expensive. Another side effect of using the system monitor is that the amount of memory consumed is increased - the DB2 Database Manager uses memory to store the data collected for every monitor element being tracked by the system monitor.

To help minimize the overhead involved in collecting system monitor information, a group of switches known as the *snapshot monitor switches* can be used to control what information is collected when a snapshot is taken; the type and amount of information collected is determined by the way these snapshot monitor switches have been set. Each snapshot monitor switch has two settings: ON and OFF. When a snapshot monitor switch is set to OFF, monitor information is not collected for elements that fall under that switch's control. The opposite is true if the switch is set to ON. (Keep in mind that a considerable amount of monitoring information is not under switch control and will always be collected regardless of how the snapshot monitor switches have been set.) The snapshot monitor switches available, along with a description of the type of information that is collected when each is set to ON, can be seen in Table 1.

Table 1. Snapshot monitor switches

Monitor Group	Monitor Switch	DBM Configuration Parameter	Information Provided
Buffer Pools	BUFFERPOOL	dft_mon_bufferpool	Amount of buffer pool activity (in other words, number of read and write operations performed and the amount of time taken for each read/write operation).
Locks	LOCK	dft_mon_lock	Number of locks held and number of deadlock cycles encountered.
Sorts	SORT	dft_mon_sort	Number of sort operations performed, number of heaps used, number of overflows encountered, and sort performance.
SQL Statements	STATEMENT	dft_mon_stmt	SQL statement processing start time, SQL statement processing end time, and SQL statement identification.

Tables	TABLE	dft_mon_table	Amount of table activity performed such as number of rows read, number of rows written, etc.
Timestamps	TIMESTAMP	dft_mon_timestamp	Times and timestamp information.
Transactions	UOW	dft_mon_uow	Transaction start times, transaction completion times, and transaction completion status.

By default, all of the switches shown in Table 1 are set to OFF, with the exception of the TIMESTAMP switch, which is set to ON and initialized when an instance is first started.

Viewing current snapshot monitor switch settings

It was mentioned earlier that the type and amount of information collected when a snapshot is taken is controlled, to some extent, by the way the snapshot monitor switches have been set. Before you take a snapshot, it is important that you know which snapshot monitor switches have been turned on and which snapshot monitor switches remain off. How can you find out what the current setting of each snapshot monitor switch available is? The easiest way is by executing the `GET MONITOR SWITCHES` command from the DB2 Command Line Processor (CLP). The basic syntax for this command is:

`GET MONITOR SWITCHES <AT DBPARTITIONNUM [`*`PartitionNum`*`]>`

where *PartitonNum* identifies the database partition (in a multi-partitioned database environment) for which the status of the snapshot monitor switches available is to be obtained and displayed.

Note : Parameters shown in angle brackets (< >) are optional; parameters or options shown in normal brackets ([ ]) are required; and a comma, followed by ellipses (...) indicate that the preceding parameter can be repeated multiple times.

If you wanted to obtain and display the status of the snapshot monitor switches for a single-partition database, you could do so by executing a `GET MONITOR SWITCHES` command that looks something like this:

```
GET MONITOR SWITCHES
```

When this command is executed from the Command Line Processor, you should see something like the output shown below.

Output from GET MONITOR SWITCHES command

```
                       Monitor Recording Switches
          Switch list for db partition number 0
Buffer Pool Activity Information (BUFFERPOOL) = OFF
Lock Information                        (LOCK) = OFF
Sorting Information                     (SORT) = OFF
SQL Statement Information          (STATEMENT) = OFF
Table Activity Information             (TABLE) = OFF
Take Timestamp Information         (TIMESTAMP) = ON   06-12-2006 10:30:00.028810
Unit of Work Information                 (UOW) = OFF
```

Upon close examination of this output, notice that the TIMESTAMP snapshot monitoring switch has been turned on and that all other switches are off. The timestamp value that follows the TIMESTAMP monitoring switch's state tells you the exact date and time the TIMESTAMP monitoring switch was turned on (which in this case is June 12, 2006, at 10:30 AM).

Changing the state of a snapshot monitor switch

Once you know which snapshot monitor switches have been turned ON and which snapshot monitor switches have been turned OFF, you may find it necessary to change one or more switch settings before you begin the monitoring process. Snapshot monitor switch settings can be changed at the instance level by modifying the appropriate DB2 Database Manager configuration parameters (see Table 1) with the `UPDATE DATABASE MANAGER CONFIGURATION` command.

On the other hand, snapshot monitor switch settings can be changed at the application level by executing the `UPDATE MONITOR SWITCHES` command. The basic syntax for this command is:

UPDATE MONITOR SWITCHES USING [[*SwitchID*] ON | OFF ,...]

where *SwitchID* identifies one or more snapshot monitor switches whose state is to be changed. This parameter may contain any or all of the following values: `BUFFERPOOL`, `LOCK`, `SORT`, `STATEMENT`, `TABLE`, `TIMESTAMP`, and `UOW`.

If you wanted to change the state of the LOCK snapshot monitor switch to ON at the application level), you could do so by executing an `UPDATE MONITOR SWITCHES` command that looks like this:

```
UPDATE MONITOR SWITCHES USING LOCKS ON
```

Likewise, if you wanted to change the state of the BUFFERPOOL snapshot monitor switch to OFF, you could do so by executing a UPDATE MONITOR SWITCHES command that looks like this:

```
UPDATE MONITOR SWITCHES USING BUFFERPOOL OFF
```

Setting snapshot monitor switches at the instance level (using the `UPDATE DATABASE MANAGER CONFIGURATION` command) affects all databases under the instance's control (in other words, every application that establishes a connection to a database under the instance's control will inherit the switch settings made in the instance's configuration). Additionally, snapshot monitor switch settings made at the instance level remain persistent across instance restarts.

Setting monitor switches at the application level (using the `UPDATE MONITOR SWITCHES` command) only affects the database a single application is interacting with. In addition, switch setting made are only persistent for the life of the application.

Capturing snapshot monitor data

As soon as a database is activated or a connection to a database is established, the snapshot monitor begins collecting monitor data. However, before any data collected can be viewed, a snapshot must be taken. (A snapshot is essentially a picture of what the monitor elements being used look like at a specific point in time.) Snapshots can be taken by embedding the `db2GetSnapshot()` API in an application program, or by executing the `GET SNAPSHOT` command. The basic syntax for this command is:

```
GET SNAPSHOT FOR
[DATABASE MANAGER | DB MANAGER | DBM] |
ALL DATABASES |
ALL APPLICATIONS |
ALL BUFFERPOOLS |
ALL REMOTE_DATABASES |
ALL REMOTE_APPLICATIONS |
ALL ON [DatabaseAlias] |
DATABASE ON [DatabaseAlias] |
APPLICATIONS ON [DatabaseAlias] |
TABLES ON [DatabaseAlias] |
TABLESPACES ON [DatabaseAlias] |
LOCKS ON [DatabaseAlias] |
BUFFERPOOLS ON [DatabaseAlias] |
DYNAMIC SQL ON [DatabaseAlias]
```

where *DatabaseAlias* identifies the alias assigned to the database that snapshot monitor information is to be collected for.

If you want to take a snapshot that only contains data collected on locks being held by applications interacting with a database named PAYROLL, you could do so by executing the following command:

```
GET SNAPSHOT FOR LOCKS ON PAYROLL
```

The output produced by this command would look something like that shown in below. (Keep in mind that this is a simple example. A real monitoring situation usually generates a large amount of data.)

Sample output from GET SNAPSHOT command

```
          Database Lock Snapshot
```

```
Database name                              = PAYROLL
Database path                              = C:\DB2\NODE0000\SQL00002\
Input database alias                       = PAYROLL
Locks held                                 = 2
Applications currently connected           = 1
Agents currently waiting on locks          = 0
Snapshot timestamp                         = 06-12-2004 08:39:40.750316

Application handle                         = 8
Application ID                             = *LOCAL.DB2.00E286133931
Sequence number                            = 0001
Application name                           = db2bp.exe
CONNECT Authorization ID                   = DB2ADMIN
Application status                         = UOW Waiting
Status change time                         = Not Collected
Application code page                      = 1252
Locks held                                 = 2
Total wait time (ms)                       = Not Collected

List Of Locks
 Lock Name                   = 0x94928D848F9F949E7B89505241
 Lock Attributes             = 0x00000000
 Release Flags               = 0x40000000
 Lock Count                  = 1
 Hold Count                  = 0
 Lock Object Name            = 0
 Object Type                 = Internal P Lock
 Mode                        = S

 Lock Name                   = 0x96A09A989DA09A7D8E8A6C7441
 Lock Attributes             = 0x00000000
 Release Flags               = 0x40000000
 Lock Count                  = 1
 Hold Count                  = 0
 Lock Object Name            = 0
 Object Type                 = Internal P Lock
 Mode                        = S
```

As you can see, the `GET SNAPSHOT` command can be used to capture several different types of monitoring data, including:

- DB2 Database Manager instance data
- Database data for all active databases under an instance's control
- Application data
- Buffer pool activity data
- Tablespace data
- Table data
- Lock data (information about all locks held)
- Dynamic SQL data (point-in-time information about SQL statements being held in the SQL statement cache)

You may also have noticed that there is a direct correlation between the snapshot monitor switches available and the different types of monitoring data that can be collected when a snapshot is taken. If a particular snapshot monitor switch is turned off and a snapshot of the monitoring elements associated with that switch is taken, the monitoring data captured may not contain any values at all. (In the previous example, some values were listed as Not Collected because the corresponding snapshot monitor switch was turned off. Furthermore, if no locks had been acquired at the time the snapshot was taken, the value for Locks held would have been 0 and the List of Locks information shown would not have been produced.)

Capturing snapshot monitor data using SQL

With earlier versions of DB2 UDB, the only way to capture snapshot monitor data was by executing the `GET SNAPSHOT` command or by calling its corresponding API from an application program. With DB2 UDB version 8.1, the ability to capture snapshot monitor data by constructing a query was introduced. This method relied on twenty special snapshot monitor table functions that have been depreciated in version 9.1. Now, snapshot monitor data can be obtained by using a new set of SQL routines to access data stored in special administrative views. These routines and views are described in Table 2.

Table 2. Snapshot administrative SQL routines and views

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

		contains information about the longest running SQL statements in the currently connected database.
QUERY_PREP_COST	N/A	This administrative view contains a list of SQL statements, along with information about the time required to prepare each statement.
N/A	SNAP_WRITE_FILE	This procedure writes system snapshot data to a file in the *tmp* subdirectory of the instance directory.
SNAPAGENT	SNAP_GET_AGENT	The administrative view and table function returns information about agents from an application snapshot, in particular, the agent logical data group.
SNAPAGENT_MEMORY_POOL	SNAP_GET_AGENT_MEMORY_POOL	This administrative view and table function returns information about memory usage at the agent level.
SNAPAPPL	SNAP_GET_APPL	The administrative view and table function returns information about applications from an application snapshot, in particular, the appl logical data group.
SNAPAPPL_INFO	SNAP_GET_APPL_INFO	The administrative view and table function returns information about applications from an application snapshot, in particular, the appl_info logical data group.
SNAPBP	SNAP_GET_BP	The administrative view and table function returns information about buffer pools from a bufferpool snapshot, in particular, the bufferpool logical data group.
SNAPBP_PART	SNAP_GET_BP_PART	The administrative view and table function returns information about buffer pools from a bufferpool snapshot, in particular, the bufferpool_nodeinfo logical data group.
SNAPCONTAINER	SNAP_GET_CONTAINER_V91	The administrative view and table function returns table space snapshot information from the tablespace_container logical data group.

SNAPDB	SNAP_GET_DB_V91	The administrative view and table function returns snapshot information from the database (dbase) and database storage (db_storage_group) logical groupings.
SNAPDB_MEMORY_POOL	SNAP_GET_DB_MEMORY_POOL	The administrative view and table function returns information about memory usage at the database level for UNIX(R) platforms only.
SNAPDBM	SNAP_GET_DMB	The administrative view and table function returns the snapshot monitor DB2 database manager (dbm) logical grouping information.
SNAPDBM_MEMORY_POOL	SNAP_GET_DBM_MEMORY_POOL	The administrative view and table function returns information about memory usage at the database manager level.
SNAPDETAILLOG	SNAP_GET_DETAILLOG_V91	The administrative view and table function returns snapshot information from the detail_log logical data group.
SNAPDYN_SQL	SNAP_GET_DYN_SQL_V91	The administrative view and table function returns snapshot information from the dynsql logical data group.
SNAPFCM	SNAP_GET_FCM	The administrative view and table function returns information about the fast communication manager (FCM) from a database manager snapshot, in particular, the fcm logical data group.
SNAPFCM_PART	SNAP_GET_FCM_PART	The administrative view and table function returns information about the fast communication manager (FCM) from a database manager snapshot, in particular, the fcm_node logical data group.
SNAPHADR	SNAP_GET_HADR	The administrative view and table function returns information about high availability disaster recovery from a database snapshot, in particular, the hadr logical data group.
SNAPLOCK	SNAP_GET_LOCK	The administrative view and table function returns snapshot information about locks, in particular, the lock logical data

		group.
SNAPLOCKWAIT	SNAP_GET_LOCKWAIT	The administrative view and table function returns snapshot information about lock waits, in particular, the lockwait logical data group.
SNAPSTMT	SNAP_GET_STMT	The administrative view and table function returns information about statements from an application snapshot.
SNAPSTORAGE_PATHS	SNAP_GET_STORAGE_PATHS	The administrative view and table function returns a list of automatic storage paths for the database including file system information for each storage path, specifically, from the db_storage_group logical data group
SNAPSUBSECTION	SNAP_GET_SUBSECTION	The administrative view and table function returns information about application subsections, namely the subsection logical monitor grouping.
SNAPSWITCHES	SNAP_GET_SWITCHES	The administrative view and table function returns information about the database snapshot switch state.
SNAPTAB	SNAP_GET_TAB_V91	The administrative view and table function returns snapshot information from the table logical data group.
SNAPTAB_REORG	SNAP_GET_TAB_REORG	The administrative view and table function return table reorganization information.
SNAPTBSP	SNAP_GET_TBSP_V91	The administrative view and table function returns snapshot information from the tablespace logical data group.
SNAPTBSP_PART	SNAP_GET_TBSP_PART_V91	The administrative view and table function returns snapshot information from the tablespace_nodeinfo logical data group.
SNAPTBSP_QUIESCER	SNAP_GET_TBSP_QUIESCER	The administrative view and table function returns information about quiescers from a table space snapshot.
SNAPTBSP_RANGE	SNAP_GET_TBSP_RANGE	The administrative view and table function returns information from a range snapshot.

SNAPUTIL	SNAP_GET_UTIL	The administrative view and table function returns snapshot information on utilities from the utility_info logical data group.
SNAPUTIL_PROGRESS	SNAP_GET_UTIL_PROGRESS	The administrative view and table function returns information about utility progress, in particular, the progress logical data group.
TBSP_UTILIZATION	N/A	This administrative view contains table space configuration and utilization information.
TOP_DYNAMIC_SQL	N/A	This administrative view contains the top dynamic SQL statements sortable by number of executions, average execution time, number of sorts, or sorts per statement.

If you wanted to obtain lock information for the currently connected database for example, you could do so by executing a query that looks something like this:

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
FROM SYSIBMADM.SNAPLOCK
```

The `SNAP_GET_LOCK` table function returns the same information as the `SNAPLOCK` administrative view, but allows you to retrieve the information for a specific database or a specific database on a specific database partition (instead of the current connected database). A query using the `SNAP_GET_LOCK` table function would look something like this:

```
SELECT AGENT_ID, LOCK_OBJECT_TYPE, LOCK_MODE, LOCK_STATUS
FROM TABLE(SNAP_GET_LOCK('',-1)) AS T
```

When used with the `SNAP_GET_LOCKWAIT` table function, the `SNAP_GET_LOCK` table function provides information equivalent to the `GET SNAPSHOT FOR LOCKS ON [`*`DatabaseAlias`*`]` command.

Resetting snapshot monitor counters

Earlier, you saw that one of the element types that monitor elements use to store data is a counter and that counters keep a running total of the number of times an activity or event occurs. Counter values increase throughout the life of the monitor. So when exactly does counting begin? Counting typically begins as soon as a snapshot monitor switch is turned on or when connection to a database is established (if instance level monitoring is used, counting begins the first time an application establishes a connection to a database under the instance's control).

However, there may be times when it is desirable to reset all counters to zero without turning snapshot monitor switches off and back on and without terminating and reestablishing database connections. By far the easiest way to quickly reset all snapshot monitor counters to zero is by executing the `RESET MONITOR` command. The basic syntax for this command is:

```
RESET MONITOR ALL
```

or

```
RESET MONITOR FOR [DATABASE | DB] [DatabaseAlias]
```

where *DatabaseAlias* identifies the alias assigned to the database that snapshot monitor counters are to be reset for.

If you wanted to reset the snapshot monitor counters for all databases under an instance's control to zero, you could do so by attaching to that instance and executing a `RESET MONITOR` command that looks like this:

```
RESET MONITOR ALL
```

On the other hand, if you wanted to reset just the snapshot monitor counters associated with a database named SAMPLE to zero, you could do so by executing a `RESET MONITOR` command that looks like this:

```
RESET MONITOR FOR DATABASE SAMPLE
```

It is important to note that you cannot selectively reset counters for a particular monitoring group that is controlled by a snapshot monitor switch using the `RESET MONITOR` command. To perform this type of operation, you must turn the appropriate snapshot monitor switch off and back on or terminate and reestablish database connections.

Section 3. Capturing event monitor data

Event monitors

You have just seen that the snapshot monitor provides a way to capture and record information about the state of an instance or a database at a specific point in time. In contrast, event monitors collect monitor data as specific events or transitions occur. Event monitors provide a way to collect monitor data when events or activities occur that cannot be monitored using the snapshot monitor.

For example, suppose you want to capture monitor data whenever a deadlock cycle occurs. If you're familiar with the concept of deadlocks, you know that a special process known as the deadlock detector (daemon) runs quietly in the background and "wakes up" at predefined intervals to scan the locking system for deadlock cycles. If a deadlock cycle is found, the deadlock detector randomly selects, rolls back, and terminates one of the transactions involved in the cycle. As a result, the selected transaction receives an SQL error code, and all locks acquired on its behalf are released so that the remaining transactions can proceed. Information about such a series of events cannot be captured by the snapshot monitor because, in all likelihood, the deadlock cycle will have been broken long before a snapshot can be taken. An event monitor, on the other hand, could capture important information about such an event because it would be activated the moment the deadlock cycle was detected.

There is another significant difference between these two monitors - the snapshot monitor exists as a background process that begins capturing monitor data once a connection to a database has been established. In contrast, event monitors must be specifically created before they can be used. Several different event monitors can exist, and each event monitor is activated only when a specific type of event or transition occurs. Table 3 shows the types of events that can cause an event monitor to be activated, along with the kind of monitor data that is collected for each event type.

Table 3. Event types and the data collected for each

Event Type	Data Collected	When Data Is Collected	Associated Group (Target Table) Names
DATABASE	The values of all database-level counters	When the database is deactivated, or when the last application connected to the database disconnects	DB, CONTROL
BUFFERPOOLS	The values of all buffer pool counters, prefetchers, and page cleaners, as well as direct I/O for each buffer pool used	When the database is deactivated or when the last application connected to the database disconnects	BUFFERPOOL, CONTROL
TABLESPACES	The values of all buffer pool counters, prefetchers, page cleaners, as well as direct I/O for each tablespace used	When the database is deactivated or when the last application connected to the database disconnects	TABLESPACE, CONTROL
TABLES	The number of rows read and the number of rows written for each table	When the database is deactivated or when the last application connected to the database disconnects	TABLE, CONTROL
DEADLOCKS	Comprehensive	When a deadlock cycle	CONNHEADER,

	information regarding applications involved, including the identification of all SQL statements involved (along with statement text) and a list of locks held by each	is detected	DEADLOCK, DLCONN, DLLOCK, CONTROL
CONNECTIONS	The values of all application-level counters	When an application that is connected to the database disconnects	CONNHEADER, CONN, CONTROL
STATEMENTS	Statement start/stop time, amount of CPU used, text of dynamic SQL statements, SQLCA (the return code of the SQL statement), and other metrics such as fetch count. For partitioned databases: amount of CPU used, execution time, table information, and table queue information	When an SQL statement finishes executing, or, for partitioned databases, when a subsection of an SQL statement finishes executing	CONNHEADER, STMT, SUBSECTION, CONTROL
TRANSACTIONS	Transaction start/stop time, previous transaction time, amount of CPU consumed, along with locking and logging metrics (transaction records aren't generated if the database uses two-phase commit processing and an X/Open XA Interface)	When a transaction is terminated (by a COMMIT or ROLLBACK statement)	CONNHEADER, XACT, CONTROL

Because event monitors are special database objects that must be created before they can be used, they can only collect monitor data for events or transitions that take place in the database for which they have been defined. Event monitors cannot be used to collect monitor data at the instance level.

Creating event monitors

You can create event monitors directly from the Control Center (select Create Event Monitor from the Event Monitors menu) or by executing the `CREATE EVENT MONITOR` SQL statement. The basic syntax for this statement is:

```
 CREATE EVENT MONITOR [Name]
FOR [DATABASE | BUFFERPOOLS | TABLESPACES | TABLES | DEADLOCKS <WITH DETAIL> |
    CONNECTIONS <WHERE [EventCondition]> |
    STATEMENTS <WHERE [EventCondition]]>  |
    TRANSACTIONS <WHERE [EventCondition]> , ...]
WRITE TO [TABLE [GroupName] (TABLE [TableName]) | PIPE [PipeName] | FILE [DirectoryName]]
[MANUALSTART | AUTOSTART]
```

where

- *Name* identifies the name to be assigned to the event monitor being created.
- *EventCondition* identifies a condition used to determine which CONNECTION, STATEMENT, or TRANSACTION the event monitor collects data for.
- *GroupName* identifies the logical data group for which the target table is defined. (See Table 3 for the appropriate values to use for this parameter.)
- *TableName* identifies the name assigned to the database table that all event monitor data is to be written to.
- *PipeName* identifies the name assigned to the named pipe that all event monitor data is to be written to.
- *DirectoryName* identifies the name assigned to the directory that one or more files containing event monitor data is be written to.

Let's say you want to create an event monitor that captures the values of all application-level counters and writes them to a database table named CONN_DATA every time an application terminates its connection to a database. To do that, execute a `CREATE EVENT MONITOR` statement that looks something like this:

```
CREATE EVENT MONITOR CONN_EVENTS FOR CONNECTIONS WRITE TO TABLE CONN (TABLE CONN_DATA)
```

Now let's say you want to create an event monitor that captures monitor data for both buffer pool and tablespace events and writes all data collected to a directory named */export/home/bpts_data*. To do that, execute a `CREATE EVENT MONITOR` statement that looks something like this:

```
CREATE EVENT MONITOR BPTS_EVENTS FOR BUFFERPOOLS, TABLESPACES WRITE TO FILE
  '/export/home/BPTS_DATA'
```

As you can see, when creating an event monitor you must specify the type of event that will cause the event monitor to be activated, as well as the location where all data collected is to be written.

Output from an event monitor can be written to one or more database tables, one or more external files, or a named pipe. Table and pipe event monitors stream event records directly to the table or named pipe specified. File event monitors, on the other hand, stream event records to a series of eight-character numbered files that have the extension `.evt` (for example, `00000000.evt`, `00000001.evt`, and so on). The data stored in these files should be treated as if it were a single data stream stored in a single file, even though the data is actually broken up into several small pieces (the start of the data stream is the first byte found in the file named `00000000.evt` and the end of the data stream is the last byte found in the last file created).

If you specify that output from an event monitor will be stored in database tables, all target tables are automatically created when the `CREATE EVENT MONITOR` statement is executed. (If the creation of a table fails for any reason, an error code is generated and the `CREATE EVENT MONITOR` statement fails.) However, if you specify that output from an event monitor will be written to one or more external files or a named pipe, the output directory or named pipe specified must exist and the DB2 Database Manager instance owner must be able to write to it at the time the event monitor is activated. Additionally, if a named pipe is used, the application monitoring the named pipe must be running and it must have opened the pipe for reading before the event monitor is activated.

Starting and stopping event monitors

If you specify the `AUTOSTART` option when creating an event monitor, the monitor will start automatically when the database containing the event monitor is started. (A database is started when it is activated with the `ACTIVATE DATABASE` command or when the first connection to the database is established.) If you use the `MANUALSTART` option or don't specify either option (in which case, `MANUALSTART` is used by default), the resulting event monitor won't collect monitor data until it has been started. Event monitors can be started (and stopped) by executing the `SET EVENT MONITOR` SQL statement. The basic syntax for this statement is:

```
SET EVENT MONITOR [MonitorName] STATE <=> [MonitorState]
```

where *MonitorName* identifies the name of the event monitor whose state is to be altered and *MonitorState* identifies the state the specified event monitor is to be placed in. To start an event monitor (in other words, place it in the 'active' state), you must specify the value **1** for the *MonitorState* parameter. To stop an event monitor (in other words, place it in the 'inactive' state), specify the value **0**.

Let's say you want to start an event monitor named CONN_EVENTS that was created with the `MANUALSTART` option. Do that by executing the following statement:

```
SET EVENT MONITOR CONN_EVENTS STATE 1
```

On the other hand, if you want to stop the CONN_EVENTS event monitor, execute a statement that looks like this:

```
SET EVENT MONITOR CONN_EVENTS STATE 0
```

You can also start and stop event monitors by highlighting the appropriate event monitor name in the Control Center and selecting an action (Start Event Monitoring or Stop Event Monitoring) from the Event Monitors menu.

The SQL function `EVENT_MON_STATE` can be used to determine the current state of any event monitor that has been defined for a database. This function must be used in a query that looks something like this:

```
SELECT EVENT_MON_STATE('CONN_EVENTS') FROM SYSIBM.SYSDUMMY1
```

(In this example, the table SYSIBM.SYSDUMMY1 is an empty table that is commonly used as a placeholder.)

Once started, an event monitor sits quietly in the background and waits for one of the events or transitions it's designed to monitor to take place. When such an event or transition occurs, the event monitor collects the appropriate monitor data and writes it to the monitor's output target (table, directory, or named pipe). The event or transition itself controls when monitor data is collected; the DBA doesn't need to perform any additional steps (unlike when the snapshot monitor is used).

Forcing an event monitor to generate output

At times, an event monitor that has a low record-generation frequency (such as one designed to monitor DATABASE events) can contain event monitor data in memory that hasn't been written to the event monitor's target location yet (because only a partial event record exists). To examine the contents of an event monitor's active internal buffers, execute the `FLUSH EVENT MONITOR` SQL statement. The basic syntax for this statement is:

```
FLUSH EVENT MONITOR [MonitorName] <BUFFER>
```

where *MonitorName* identifies the event monitor (by name) that you want to force to write the contents of its active internal buffers to its target location.

To force an event monitor named CONN_EVENTS to write the contents of its active internal buffers to its target location, execute a `FLUSH EVENT MONITOR` statement that looks like this:

```
FLUSH EVENT MONITOR CONN_EVENTS
```

By default, records that are written to an event monitor's target location prematurely are logged in the event monitor log and assigned a partial record identifier. However, if you specify the `BUFFER` option when executing the `FLUSH EVENT MONITOR` statement, only monitor data present in the event monitor's active internal buffers is written to the event monitor's target location. No partial record is logged in the event monitor log.

It is important to note that when event monitors are flushed, counters aren't reset. As a result, the event monitor record that would have been generated had the `FLUSH EVENT MONITOR` statement not been executed will still be generated when the event monitor is triggered normally.

Viewing event monitor data

Earlier, you saw that event monitor data can be written to one of three different locations:

1. **Files.** Event monitor output can be written to one or more files. Two parameters control the amount of space available for use (MAXFILESIZE and MAXFILES) and once the space limit is reached the event monitor will automatically flush all events and stop itself. The default setting for both parameters is NONE, which indicates that there is no space limit.

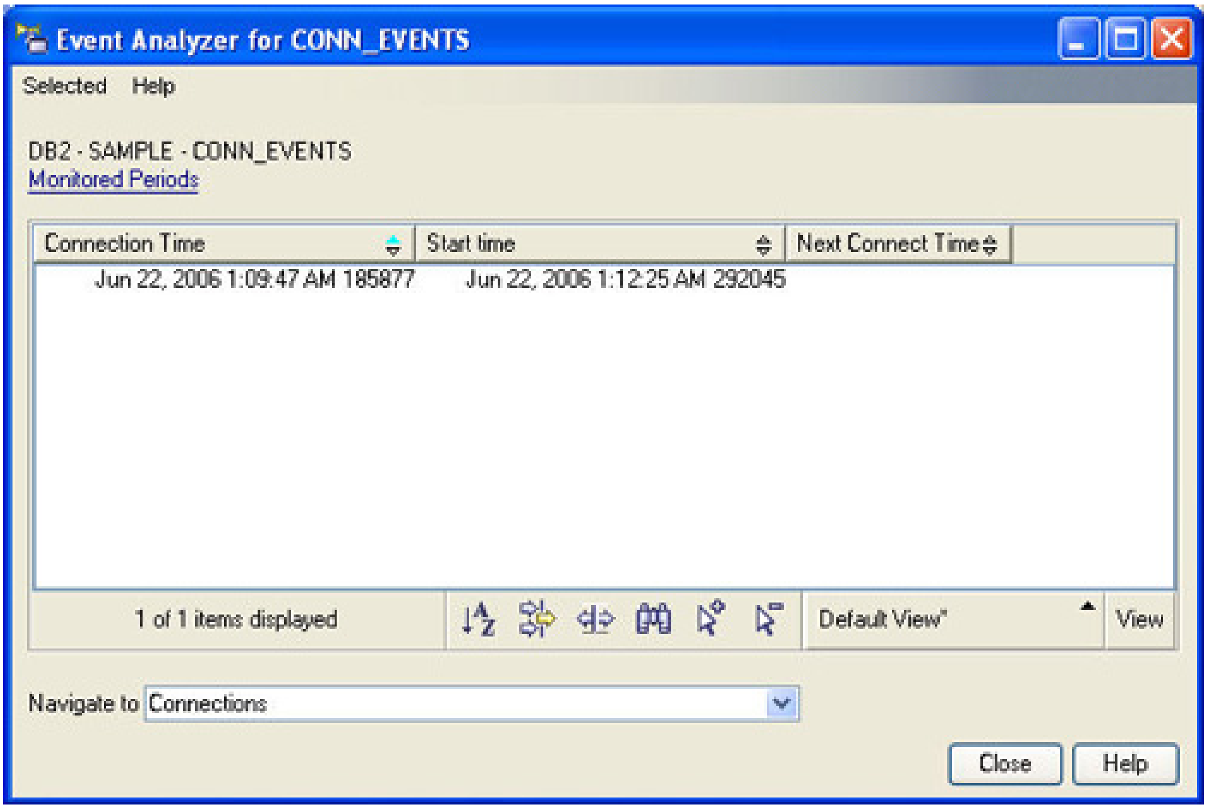
2. **Pipes.** Event monitor output can be written to a named pipe. The name of the pipe must be provided, but the named pipe itself does not need to exist when the event monitor is created. It must, however, exist when the event monitor is activated.

3. **Tables.** Event monitor output can be written out to one or more tables that exist in the database. Each monitor element in the event monitor is mapped to a table of the same name. Data for each individual event is inserted into the appropriate table as a single row.

At some point, you'll want to examine the data an event monitor has collected. If the data collected is written to a named pipe, the application at the receiving end of the pipe is usually responsible for displaying monitor data as it's received. If the data collected is written to a table or a set of files, you can view that data by using one of two special utilities: the Event Analyzer and the event monitor productivity tool.

The Event Analyzer is a GUI tool that can be activated by highlighting the desired event monitor in the Control Center and selecting the appropriate action from the Event Monitors menu or by executing the command `db2eva`. Once activated, the Event Analyzer lets you drill down and view information that a specific event monitor captured. Figure 1 shows what the Event Analyzer typically looks like when it is first activated.

Figure 1. The Event Analyzer



Connection Time	Start time	Next Connect Time
Jun 22, 2006 1:09:47 AM 185877	Jun 22, 2006 1:12:25 AM 292045	

1 of 1 items displayed

Default View*

Navigate to: Connections

The Event Analyzer can only be used to view event monitor data that was collected and stored in database tables. To view event monitor data that was written to files (and named pipes), you must use the text-based event monitor productivity tool, which retrieves information from an event monitor data file or named pipe and generates a formatted report. (Event monitor files and named pipes contain a binary stream of logical data groupings that must be formatted before they can be displayed.)

To activate the event monitor productivity tool, execute the `db2evmon` command. The basic syntax for this command looks like this:

```
db2evmon -db [DatabaseAlias] -evm [MonitorName]
```

where *DatabaseAlias* identifies the database (by alias) on which the event monitor whose data is to be displayed is defined and *MonitorName* identifies the name assigned to the event monitor whose data is to be displayed.

or

```
db2evmon -path [MonitorTarget]
```

where *MonitorTarget* identifies the location (directory or named pipe) where data that has been collected by the event monitor specified is stored.

For example, to format and display all data collected by an event monitor named

CONN_EVENTS that was defined in a database named SAMPLE, execute the following command (assuming monitor data was written to a file):

```
db2evmon -db SAMPLE -evm CONN_EVENTS
```

Assuming the event monitor named CONN_EVENTS was created by executing the following SQL statement:

```
CREATE EVENT MONITOR CONN_EVENTS FOR CONNECTIONS WRITE TO FILE
'C:\MONDATA' AUTOSTART
```

And assuming an application has established a connection to the SAMPLE database (which caused the event monitor to capture and produce monitoring data), the output produced when the event monitor productivity tool is used to examine that data should look something like the sample below.

Output sample produced by the event monitor productivity tool

```
Reading C:\mondata\00000000.EVT ...

--------------------------------------------------------------------------
                            EVENT LOG HEADER
  Event Monitor name: CONN_EVENTS
  Server Product ID: SQL09000
  Version of event monitor data: 8
  Byte order: LITTLE ENDIAN
  Number of nodes in db2 instance: 1
  Codepage of database: 1208
  Territory code of database: 1
  Server instance name: DB2
--------------------------------------------------------------------------

--------------------------------------------------------------------------
  Database Name: SAMPLE
  Database Path: C:\DB2\NODE0000\SQL00002\
  First connection timestamp: 06/22/2006 00:56:40.086671
  Event Monitor Start time:   06/22/2006 00:56:40.662668
--------------------------------------------------------------------------

3) Connection Header Event ...
  Appl Handle: 55
  Appl Id: *LOCAL.DB2.060622045634
  Appl Seq number: 00001
  DRDA AS Correlation Token: *LOCAL.DB2.060622045634
  Program Name    : db2bp.exe
  Authorization Id: RSANDERS
  Execution Id    : RSANDERS
  Codepage Id: 1252
  Territory code: 0
  Client Process Id: 1992
  Client Database Alias: SAMPLE
  Client Product Id: SQL09000
  Client Platform: Unknown
  Client Communication Protocol: Local
  Client Network Name: rsanders-lxp
  Connect timestamp: 06/22/2006 00:56:40.086671

4) Connection Event
...

--------------------------------------------------------------------------
  Database Name: SAMPLE
  Database Path: C:\DB2\NODE0000\SQL00002\
  First connection timestamp: 06/22/2006 00:57:36.727014
  Event Monitor Start time:   06/22/2006 00:57:37.223404
```

```
-----------------------------------------------------------------------
```

Guidelines for using event monitors

Event monitors should only be used to monitor specific events or short workloads. They are designed to provide very specific information that can be used to diagnose problems or undesired behavior of a database/application.

Unlike snapshots, event monitors have an extremely heavy impact on performance. This is due to the amount of information that is written out for each event object. Additionally, SQL statement event monitors cause an even heavier performance impact because of all the extra work the database engine has to perform each time a query is executed: instead of being able to simply execute a query, the DB2 Database Manager must also generate and record all the characteristics and runtime information associated with the query. If this information is written to a text file, that slows things down even further.

While on the subject of files, when creating event monitors that write data to files, it is a good idea to impose file size limits to control the amount of disk space that event monitor output will consume. Otherwise, if you are monitoring a high-volume OLTP system, the output can quickly grow to hundreds of megabytes.

One of the most common uses for event monitors is to capture deadlock information. (A deadlock event monitor does not write out a lot of data and is triggered sporadically so it is acceptable not to impose a file size limit.) If an event monitor is not used, it is almost impossible to determine exactly what locks and applications were involved in a deadlock cycle. A deadlock event monitor will collect information on all the applications and their locks when a deadlock cycle occurs. Armed with this information, the precise SQL statement that caused the deadlock cycle can be monitored or altered to correct the situation. Don't forget that the application that DB2 labels as the cause of a deadlock is the last application involved in the deadlock cycle - the real cause may actually be a transaction that was started much earlier by another application. Make sure you examine all the locks and applications involved to corectly determine where the problem originated.

The second most common use for event monitors is to keep track of SQL statement processing. An SQL statement event monitor can be quite useful because it traps both dynamic and static SQL statements. This is essential if an application makes use of precompiled SQL statements that would not be captured using an SQL snapshot. When an event monitor is used to capture information about every SQL statement that is executed, the properties of each statement, such as the number of rows read, selected, deleted, etc., is recorded, and is not presented as an aggregate total as is the case when a snapshot is captured. Furthermore, because the execution timeframe and start and stop times are recorded as well, detailed analysis of transactions and of how the execution of SQL by one application affects the execution of SQL by others can be performed. However, because of the volume of information produced and performance overhead required to run an SQL statement monitor, such a monitor should only be used for short tests or problem determination, and not in a production environment.

Section 4. Health monitoring and the Health Center

Health monitoring

Although the snapshot monitor and event monitors work differently (the snapshot monitor is used to capture information about the current state of an instance and/or database at a given point; event monitors are used to collect monitor data as specific events or transitions occur), they have one thing in common - both are designed to help pinpoint problem areas that are *already* adversely affecting a database system's performance. With DB2 UDB version 8.1, IBM introduced a new tool to help database administrators monitor the health of a DB2 UDB system: the *health monitor*. This tool adds a *management by exception* capability to DB2 9 by alerting administrators to potential system health issues *before* they become problems that affect a system's performance.

The health monitor reverses the system health diagnosis model from that of a DBA hunting for the source of existing problems by running snapshot and event monitors at different times and analyzing huge amounts of data looking for indications of a system's unhealthiness to DB2 monitoring itself for healthiness and notifying select personnel only when potential or existing unhealthy conditions are encountered.

How the health monitor works

The health monitor is a server-side tool that runs quietly in the background and constantly monitors the health of both a DB2 Database Manager instance and any databases that fall under its control. Unlike the database system monitor, whose use introduces additional processing overhead, the health monitor takes advantage of new monitoring technology that has no significant impact on performance. And, the health monitor requires no user intervention (another difference between it and the database system monitor).

The health monitor uses several *health indicators* to evaluate specific aspects of instance and database performance. Each health indicator acts as a precise measurement that the health monitor examines continuously to gauge the health of a particular aspect of a specific class of database objects. In turn, health indicators measure a finite set of distinct object states or a continuous range of values to determine whether an particular object is "healthy" or "unhealthy". Health indicators have a set of predefined threshold values, and the health monitor constantly compares the state of the system against these thresholds - you can modify these threshold values to meet your specific needs. If the health monitor finds that a particular threshold limit has been exceeded (for example, the amount of log space available drops below a certain level) or detects an abnormal state for a particular object (for example, an instance is down), it automatically issues an alert through the specified reporting channels. Health indicators exist for the following components:

- Instance
- Database
- Logs
- Table space storage
- Sorting
- Package and catalog caches
- Workspaces
- Memory
- Application concurrency

The health monitor can generate three types of alerts: *attention*, *warning*, and *alarm*. Health indicators that measure distinct states will issue an alert whenever a non-normal state is registered; health indicators that measure a continuous range of values use threshold values to define boundaries (or zones) for normal, attention, warning, and alarm states. For example, if a health indicator value enters the threshold range that defines an alarm zone, an alarm alert is issued to indicate that the problem needs immediate attention.

Any time an alert is raised, the health monitor may take any of the following actions to report it:

- Record the alert information in the Journal (all alarm alerts are written to the Journal)
- Send alert notifications via e-mail or a pager address to the person responsible for the system
- Carry out one or more preconfigured actions (for example, running a task)

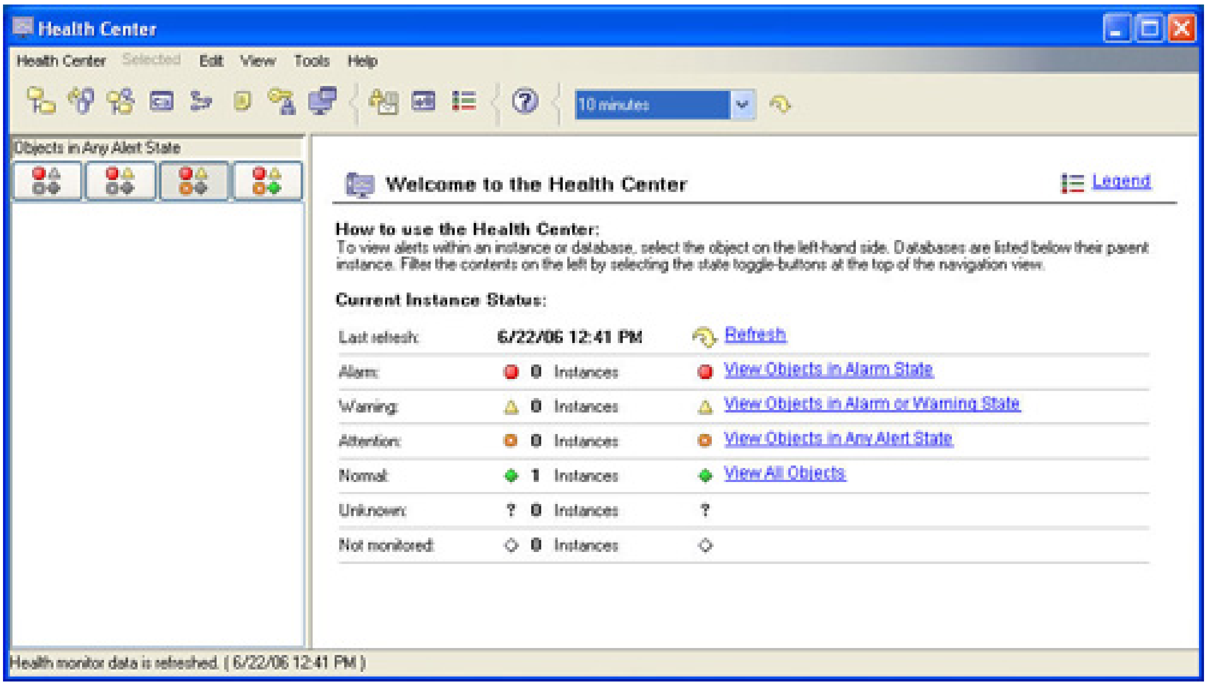
By default, the health monitor is disabled when an instance is first created. However, you can enable it at any time by selecting the appropriate menu item from the Health Center or by assigning the value **ON** to the *health_mon* DB2 Database Manager configuration parameter.

It is important to note that once the health monitor has been activated, if it generates an alert while any DB2 interface tools are active, the user is signaled using a *Health Beacon*. A Health Beacon is simply a button icon that appears on the status line of a window or notebook - by clicking on a Health Beacon, control is immediately transferred to the Health Center (which you will look at next), where you can get additional information about the alert (and recommendations for resolving the situation that caused the alert to be generated).

The Health Center

The Health Center is a GUI tool that is designed to interact with the health monitor. Figure 2 shows how the Health Center looks when it is first activated on a Windows XP system (in this case, no alerts have been generated).

Figure 2. The Health Center



Welcome to the Health Center

How to use the Health Center:
To view alerts within an instance or database, select the object on the left-hand side. Databases are listed below their parent instance. Filter the contents on the left by selecting the state toggle-buttons at the top of the navigation view.

Current Instance Status:

Last refresh:	6/22/06 12:41 PM	Refresh
Alarm:	● 0 Instances	View Objects in Alarm State
Warning:	△ 0 Instances	View Objects in Alarm or Warning State
Attention:	● 0 Instances	View Objects in Any Alert State
Normal:	◆ 1 Instances	View All Objects
Unknown:	? 0 Instances	?
Not monitored:	◇ 0 Instances	◇

Health monitor data is refreshed. (6/22/06 12:41 PM)

Like many of the GUI tools provided with DB2 9, the Health Center is composed of an *objects pane* (on the left side of the Health Center screen) and a *contents pane* (on the right side of the Health Center). These panes, show various information about system health, including:

- **The status of the database environment.** An icon is presented beside each object displayed in the objects pane that identifies the highest alert level generated for the object (or for any objects managed by that object). A green diamond icon beside an object means that the object and any objects under its control haven't raised any alerts. You can use the toggle buttons at the top of the objects pane to filter alerts according to their severity.
- **Alerts generated for an instance or a database.** When you select an object in the navigation tree in the objects pane, the alerts for that object are shown in the contents pane to the right.
- **Detailed alert information and recommended actions.** When you double-click on any alert shown in the contents pane, a notebook containing detailed information about the alert appears. The first page of this notebook contains details about the alert; the second page contains a list of recommended actions to follow to resolve the alert. In most cases, you can perform one of the recommended actions directly from the

notebook. For example, if the recommended action is to make a change to the DB2 Database Manager or database configuration, the Heath Center shows the new configuration value along with a button you can click to make the change. In other cases, the Heath Center might recommend investigating the problem further by launching a tool, such as the Command Line Processor or the Memory Visualizer.

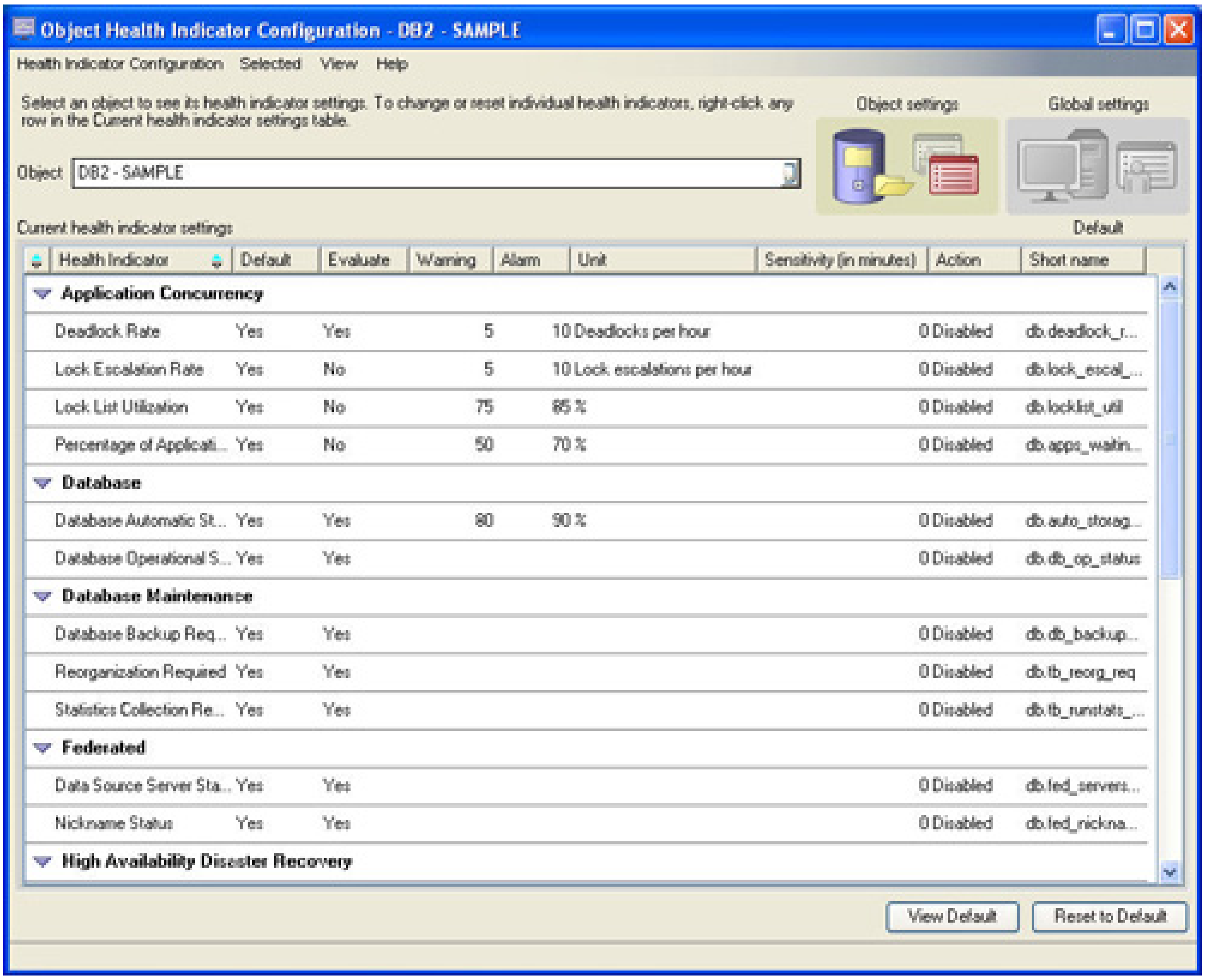
You can activate the Health Center at any time by selecting the Health Center action from the Tools menu of another DB2 9 GUI tool, by selecting the appropriate icon from the toolbar of another DB2 9 GUI tool, or by executing the command `db2hc` from the Command Line Processor. On Windows systems, you can also activate the Health Center by clicking the **Start** button and selecting **Start > Programs > IBM DB2 > Monitoring Tools > Health Center**.

Note: DB2 9 contains a tool called the *Web Health Center*, which includes all Health Center capabilities and adds the ability to access health monitor information directly from a Web browser or PDA.

Activating health indicators and defining event actions

From the Health Center, you can select the instance and database objects that you want to monitor, customize the threshold settings of any health indicator, specify where notifications are to be sent, and define what actions are to be taken if an alert is issued. By default, most health indicatiors are inactivate when the health monitor is installed. To activate a health indicator or alter a health indicator's threshold values, you must first access the Object health Indicator Configuration window of the Health Center, shown in Figure 3.

Figure 3. The Object Health Indicator Configuration window


Object Health Indicator Configuration - DB2 - SAMPLE
Health Indicator Configuration
Selected
View
Help
Object settings
Global settings
Default
View Default
Reset to Default

Select an object to see its health indicator settings. To change or reset individual health indicators, right-click any row in the Current health indicator settings table.

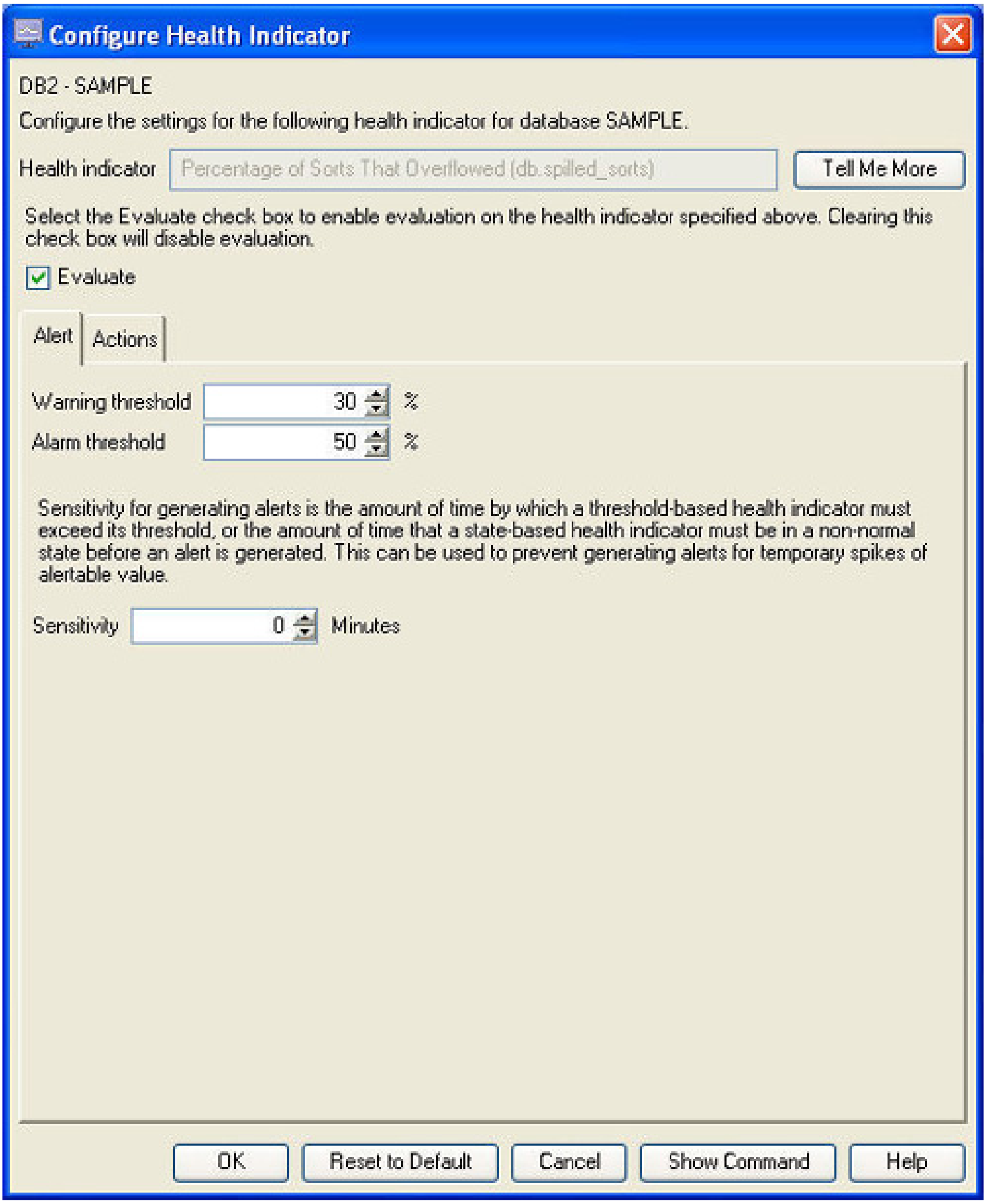
Object: DB2 - SAMPLE

Current health indicator settings

Health Indicator	Default	Evaluate	Warning	Alarm	Unit	Sensitivity (in minutes)	Action	Short name
Application Concurrency								
Deadlock Rate	Yes	Yes	5	10	Deadlocks per hour	0	Disabled	db.deadlock_r...
Lock Escalation Rate	Yes	No	5	10	Lock escalations per hour	0	Disabled	db.lock_escal_...
Lock List Utilization	Yes	No	75	85	%	0	Disabled	db.locklist_util
Percentage of Applicati...	Yes	No	50	70	%	0	Disabled	db.apps_waitin...
Database								
Database Automatic St...	Yes	Yes	80	90	%	0	Disabled	db.auto_storag...
Database Operational S...	Yes	Yes				0	Disabled	db.db_op_status
Database Maintenance								
Database Backup Req...	Yes	Yes				0	Disabled	db.db_backup...
Reorganization Required	Yes	Yes				0	Disabled	db.tb_reorg_req
Statistics Collection Re...	Yes	Yes				0	Disabled	db.tb_runstats_...
Federated								
Data Source Server Sta...	Yes	Yes				0	Disabled	db.fed_servers...
Nickname Status	Yes	Yes				0	Disabled	db.fed_nickna...
High Availability Disaster Recovery								

To activate a health indicator, you simply double-click the appropriate health indicator in the Object Health Indicator Configuration window and when the Configure Health Indicator window is displayed, shown in Figure 4, select the Evaluate check box.

Figure 4. The Configure Health Indicator window



DB2 - SAMPLE

Configure the settings for the following health indicator for database SAMPLE.

Health indicator: Percentage of Sorts That Overflowed (db.spilled_sorts)

Select the Evaluate check box to enable evaluation on the health indicator specified above. Clearing this check box will disable evaluation.

☑ Evaluate

Warning threshold: 30 %

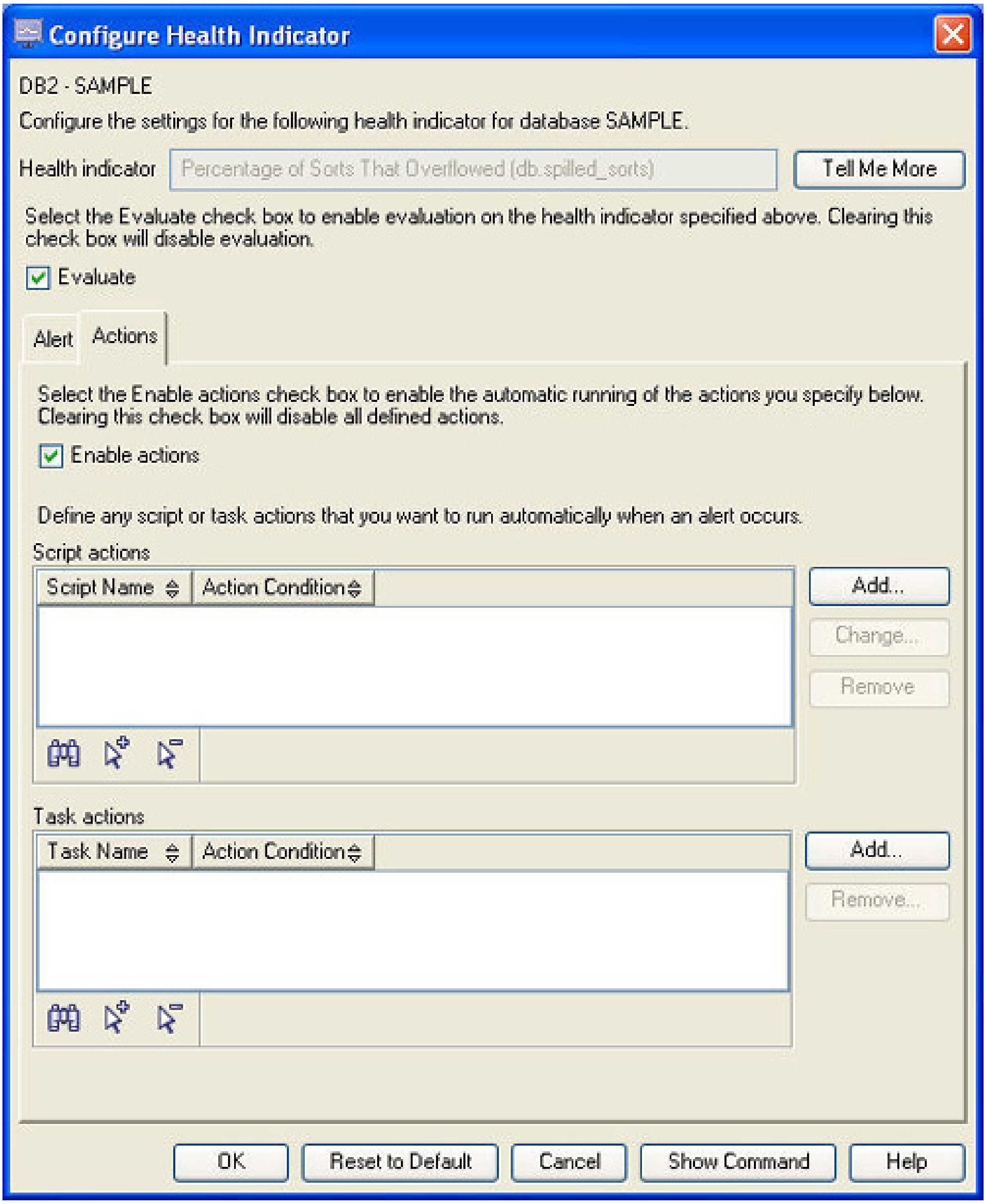
Alarm threshold: 50 %

Sensitivity for generating alerts is the amount of time by which a threshold-based health indicator must exceed its threshold, or the amount of time that a state-based health indicator must be in a non-normal state before an alert is generated. This can be used to prevent generating alerts for temporary spikes of alertable value.

Sensitivity: 0 Minutes

Using the Configure Health Indicator window, you can also specify what actions are to be taken when threshold limits for a health indicator are exceeded. Figure 5 shows the Configure Health Indicator window and the fields used to define the actions that are to be taken when different alerts are generated.

Figure 5. The Configure Health Indicator window



Configure Health Indicator

DB2 - SAMPLE

Configure the settings for the following health indicator for database SAMPLE.

Health indicator: Percentage of Sorts That Overflowed (db.spilled_sorts)

Select the Evaluate check box to enable evaluation on the health indicator specified above. Clearing this check box will disable evaluation.

☑ Evaluate

Select the Enable actions check box to enable the automatic running of the actions you specify below. Clearing this check box will disable all defined actions.

☑ Enable actions

Define any script or task actions that you want to run automatically when an alert occurs.

Script actions

Script Name	Action Condition

Task actions

Task Name	Action Condition

Section 5. Analyzing SQL with the Explain facility

What is the explain facility?

When an SQL statement is submitted to the DB2 database engine for processing, it is analyzed by the DB2 Optimizer to produce what is known as an access plan. Each access plan contains detailed information about the strategy that will be used to execute the statement (such as whether or not indexes will be used, what sort methods, if any, are required, what locks are needed, and what join methods, if any, will be used). If the SQL statement is coded in an application, the access plan is generated at precompile time (or at bind time if deferred binding is used) and an executable form of the access plan produced is stored in the system catalog as an object that is known as a package. If, however, the statement is submitted from the Command Line Processor or if the statement is a dynamic SQL statement in an application program (in other words, an SQL statement that is constructed at application run time), the access plan is generated at the time the statement is issued and the executable form produced is stored temporarily in memory (in the global package cache) rather than in the system catalog. (If an SQL statement is issued and an executable form of its access plan already exists in the global package cache, the existing access plan is reused and the DB2 Optimizer is not invoked again.)

Why is this important? Because, although the database system monitor and the health monitor can be used to obtain information about how well (or poorly) some SQL operations perform, they cannot be used to analyze individual SQL statements. To perform this type of analysis, you must be able to capture and view the information stored in an SQL statement's access plan. And in order to capture and view access plan information, you must use the DB2 9 explain facility.

The explain facility allows you to capture and view detailed information about the access plan chosen for a particular SQL statement, as well as performance information that can be used to help identify poorly written statements or a weakness in database design. Specifically, explain data helps you understand how the DB2 Database Manager accesses tables and indexes to satisfy a query. Explain data can also be used to evaluate any performance tuning action taken. In fact, any time you change some aspect of the DB2 Database Manager, an SQL statement, or the database the statement interacts with, you should collect and examine explain data to find out what effect, if any, your changes have had on performance.

Explain tables

Before explain information can be captured, a special set of tables, known as the *explain tables*, must be created. Each explain table used, along with the information it is designed to hold, can be seen in Table 4.

Table 4. The explain tables

Table Name	Contents
EXPLAIN_ARGUMENT	Contains the unique characteristics for each individual operator used, if any.
EXPLAIN_INSTANCE	Contains basic information about the source of the SQL statements being explained as well as information about the environment in which the explanation took place. (The

	EXPLAIN_INSTANCE table is the main control table for all explain information. Each row of data in the other explain tables is explicitly linked to one unique row in this table.)
EXPLAIN_OBJECT	Contains information about the data objects that are required by the access plan generated for an SQL statement.
EXPLAIN_OPERATOR	Contains all the operators that are needed by the SQL compiler to satisfy the SQL statement.
EXPLAIN_PREDICATE	Contains information that identifies which predicates are applied by a specific operator.
EXPLAIN_STATEMENT	Contains the text of the SQL statement as it exists for the different levels of explain information. The original SQL statement as entered by the user is stored in this table along with the version used by the DB2 Optimizer to choose an access plan to satisfy the SQL statement. (The latter version may bear little resemblance to the original, as it may have been rewritten and/or enhanced with additional predicates by the SQL Precompiler.)
EXPLAIN_STREAM	Contains information about the input and output data streams that exist between individual operators and data objects. (The data objects themselves are represented in the EXPLAIN_OBJECT table while the operators involved in a data stream can be found in the EXPLAIN_OPERATOR table.)

Typically, explain tables are used in a development database to aid in application design, but not in production databases where application code remains fairly static. Because of this, they are not created along with the system catalog tables as part of the database creation process. Instead, explain tables must be manually created in the database that the explain facility is to be used with before the explain facility can be used. Fortunately, the process used to create the explain tables is pretty straightforward: using the Command Line Processor, you establish a connection to the appropriate database and execute a script named EXPLAIN.DDL, which can be found in the "misc" subdirectory of the "sqllib" directory where the DB2 9 software was initially installed. (Comments in the header of this file provide information on how it is to be executed.)

Collecting explain data

The explain facility is comprised of several individual tools and not all tools require the same kind of explain data. Therefore, two different types of explain data can be collected:

- **Comprehensive explain data.** Contains detailed information about an SQL statement's access plan. This information is stored across several different explain tables.

- **Explain snapshot data.** Contains the current internal representation of an SQL statement, along with any related information. This information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT explain table.

As you might imagine, there are a variety of ways in which both types of explain data can be collected. The methods available for collecting explain data include:

- Executing the `EXPLAIN` SQL statement
- Setting the `CURRENT EXPLAIN MODE` special register
- Setting the `CURRENT EXPLAIN SNAPSHOT` special register
- Using the `EXPLAIN` bind option with the `PRECOMPILE` or `BIND` command
- Using the `EXPLSNAP` bind option with the `PRECOMPILE` or `BIND` command

EXPLAIN SQL statement

One way to collect both comprehensive explain information and explain snapshot data for a single, dynamic SQL statement is by executing the `EXPLAIN` SQL statement. The basic syntax for this statement is:

```
EXPLAIN [ALL | PLAN | PLAN SELECTION]
<FOR SNAPSHOT | WITH SNAPSHOT>
FOR [SQLStatement]
```

where *SQLStatement* identifies the SQL statement that explain data and/or explain snapshot data is to be collected for. (The statement specified must be a valid `INSERT`, `UPDATE`, `DELETE`, `SELECT`, `SELECT INTO`, `VALUES`, or `VALUES INTO` SQL statement.)

If the `FOR SNAPSHOT` option is specified with the `EXPLAIN` statement, only explain snapshot information is collected for the dynamic SQL statement specified. On the other hand, if the `WITH SNAPSHOT` option is specified instead, both comprehensive explain information and explain snapshot data is collected for the dynamic SQL statement specified. However, if neither option is used, only comprehensive explain data is collected; no explain snapshot data is produced.

To collect both comprehensive explain data and explain snapshot information for the SQL statement `SELECT * FROM DEPARTMENT`, execute an `EXPLAIN` statement that looks like this:

```
EXPLAIN ALL WITH SNAPSHOT FOR SELECT * FROM DEPARTMENT
```

On the other hand, to collect only explain snapshot data for the same SQL statement, execute an EXPLAIN statement that looks like this:

```
EXPLAIN ALL FOR SNAPSHOT FOR SELECT * FROM DEPARTMENT
```

And finally, to collect only comprehensive explain data for the SQL statement SELECT * FROM DEPARTMENT, execute an EXPLAIN statement that looks like this:

```
 EXPLAIN ALL FOR SELECT * FROM DEPARTMENT
```

It is important to note that the EXPLAIN statement does not execute the SQL statement specified, nor does it display the explain information collected. Other explain facility tools must be used to view the information collected. (We'll look at those tools shortly.)

CURRENT EXPLAIN MODE and the CURRENT EXPLAIN SNAPSHOT special registers

Although the EXPLAIN SQL statement is useful when you want to collect explain and/or explain snapshot information for a single dynamic SQL statement, it can become very time consuming to use if a large number of SQL statements need to be analyzed. A better way to collect the same information for several dynamic SQL statements is by setting one or both of the special explain facility registers provided before a group of dynamic SQL statements are executed. Then, as the statements are prepared for execution, explain and/or explain snapshot information is collected for each statement processed. (The statements themselves, however, may or may not be executed once explain and/or explain snapshot information has been collected.)

The two explain facility special registers that are used in this manner are the CURRENT EXPLAIN MODE special register and the CURRENT EXPLAIN SNAPSHOT special register. The CURRENT EXPLAIN MODE special register is set using the SET CURRENT EXPLAIN MODE SQL statement and the CURRENT EXPLAIN SNAPSHOT special register is set using the SET CURRENT EXPLAIN SNAPSHOT SQL statement. The basic syntax for the SET CURRENT EXPLAIN MODE SQL statement is:

```
SET CURRENT EXPLAIN MODE <=>
[NO |
 YES |
 EXPLAIN |
 REOPT |
 RECOMMEND INDEXES |
 EVALUATE INDEXES |
 RECOMMEND PARTITIONINGS |
 EVALUATE PARTITIONINGS]
```

The basic syntax for the SET CURRENT EXPLAIN SNAPSHOT SQL statement is:

```
SET CURRENT EXPLAIN SNAPSHOT <=> [YES | NO | EXPLAIN | REOPT]
```

As you might imagine, if both the `CURRENT EXPLAIN MODE` and the `CURRENT EXPLAIN SNAPSHOT` special registers are set to `NO`, the explain facility is disabled and no explain data is captured. On the other hand, if either special register is set to `EXPLAIN`, the explain facility is activated and comprehensive explain information or explain snapshot data (or both if both special registers have been set) is collected each time a dynamic SQL statement is prepared for execution. However, the statements themselves are not executed. If either special register is set to **YES**, the behavior is the same as when either register is set to `EXPLAIN` with one significant difference; the dynamic SQL statements that explain information is collected for are executed as soon as the appropriate explain/explain snapshot data has been collected.

If either the `CURRENT EXPLAIN MODE` or the `CURRENT EXPLAIN SNAPSHOT` special register is set to `REOPT` the explain facility is activated and explain information or explain snapshot data (or both if both special registers have been set) is captured whenever a static or dynamic SQL statement is processed during statement reoptimization at execution time; that is, when actual values for the host variables, special registers, or parameter markers used in the statement are available.

The EXPLAIN and EXPLSNAP precompile/bind options

So far, you have looked at ways in which comprehensive explain information and explain snapshot data can be collected for dynamic SQL statements. But often, database applications are comprised of static SQL statements that need to be analyzed as well. So how can you use the explain facility to analyze static SQL statements coded in an embedded SQL application? To collect comprehensive explain information and/or explain snapshot data for static and/or dynamic SQL statements that have been coded in an embedded SQL application, you rely on the `EXPLAIN` and `EXPLSNAP` precompile/bind options.

As you might imagine, the `EXPLAIN` precompile/bind option is used to control whether or not comprehensive explain data is collected for static and/or dynamic SQL statements that have been coded in an embedded SQL application. Likewise, the `EXPLSNAP` precompile/bind option controls whether or not explain snapshot data is collected. One or both of these options can be specified as part of the `PRECOMPILE` command that is used to precompile the source code file that contains the embedded SQL statements. If deferred binding is used, these options can be provided with the `BIND` command that is used to bind the application's bind file to the database.

Both the `EXPLAIN` option and the `EXPLSNAP` option can be assigned the value `NO`, `YES`, `ALL`, or `REOPT`. If both options are assigned the value `NO` (for example, `EXPLAIN NO EXPLSNAP NO`), the Explain facility is disabled and no explain data is captured. On the other hand, if either option is assigned the value `YES`, the explain facility is activated and comprehensive explain information or explain snapshot data (or both if both options are set) is collected for each static SQL statement found in the application. If either option is assigned the value `ALL`, the explain facility is activated and comprehensive explain information or explain snapshot data (or both if both options are set) is collected for every static *and every dynamic* SQL statement found, even if the `CURRENT EXPLAIN MODE` and/or the `CURRENT EXPLAIN SNAPSHOT` special registers have been set to `NO`.

If either the `EXPLAIN` or the `EXPLSNAP` option is assigned the value `REOPT` explain information or explain snapshot data (or both if both options have been specified) for each reoptimizable incremental bind SQL statement will be placed in the explain tables at run time, even if the `CURRENT EXPLAIN MODE` and/or the `CURRENT EXPLAIN SNAPSHOT` special registers have been set to `NO`.

Evaluating explain data

So far, you have concentrated on the various ways in which comprehensive explain and explain snapshot data can be collected. But once the data is collected, how can it be viewed? To answer this question, you need to take a look at the explain facility tools that have been designed specifically for presenting explain information in a meaningful format. These tools include:

- `db2expln`
- `db2exfmt`
- Visual Explain

db2expln

Earlier, you saw that when a source code file containing embedded SQL statements is bound to a database (either as part of the precompile process or during deferred binding), the DB2 Optimizer analyzes each static SQL statement encountered and generates a corresponding access plan, which is then stored in the database in the form of a package. Given the name of the database, the name of the package, the ID of the package creator, and a section number (if the section number **0** is specified, all sections of the package is processed), the `db2expln` tool interprets and describes the access plan information for any package that is stored in a database's system catalog. Since the `db2expln` tool works directly with a package and not with comprehensive explain or explain snapshot data, it is typically used to obtain information about the access plans that have been chosen for packages for which explain data has not been captured. However, because the `db2expln` tool can only access information that has been stored in a package, it can only describe the implementation of the final access plan chosen; it cannot provide information on how a particular SQL statement was optimized.

Using additional input parameters, the `db2expln` tool can also be used to explain dynamic SQL statements (that do not contain parameter markers).

db2exfmt

Unlike the `db2expln` tool, the `db2exfmt` tool is designed to work directly with comprehensive explain or explain snapshot data that has been collected and stored in the explain tables. Given a database name and other qualifying information, the `db2exfmt` tool queries the explain tables for information, format the results, and produce a text-based report that can be displayed directly on the terminal or written to an ASCII file.

Visual Explain

Visual Explain is a GUI tool that provides database administrators and application developers with the ability to view a graphical representation of the access plan that has been chosen for a particular SQL statement. In addition, Visual Explain allows you to:

- See the database statistics that were used to optimize the SQL statement.
- Determine whether or not an index was used to access table data. (If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.)
- View the effects of performance tuning by allowing you to make "before" and "after" comparisons.
- Obtain detailed information about each operation that is performed by the access plan, including the estimated cost of each.

However, Visual Explain can only be used to view explain snapshot data; to view explain data that has been collected and written to the explain tables, the `db2exfmt` tool must be used instead.

As you can see, the various tools that are available for displaying comprehensive explain information and explain snapshot data vary greatly both in their complexity in the capa-bilities they provide. Table 5 summarizes the different tools available, and highlights their individual characteristics. To get the most out of the explain facility, you should consider your environment and your needs when making a decision on which tool to use.

Table 5. Comparison of explain facility tools available

Desired Characteristics	Visual Explain	db2exfmt	db2expln
User interface	Graphical	Text-based	Text-based
"Quick and dirty" static SQL analysis	No	No	Yes
Static SQL supported	Yes	Yes	Yes
Dynamic SQL supported	Yes	Yes	Yes
CLI applications supported	Yes	Yes	No
Detailed DB2 Optimizer	Yes	Yes	No

information available			
Suited for analysis of multiple SQL statements	No	Yes	Yes

Section 6. Visual Explain - A closer look

Timerons and SQL translation

Timerons

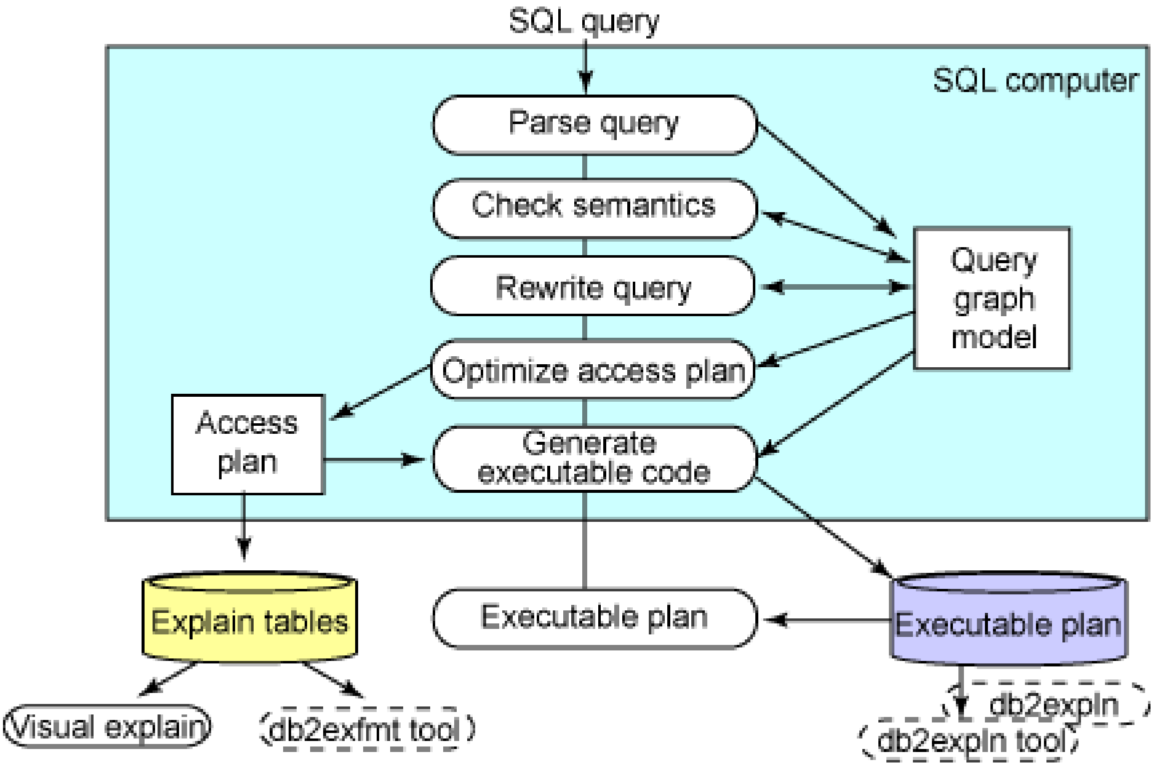
The most important thing that you need understand in order to analyze explain information is the concept of the *timeron.* A timeron is a unit of measurement used by the DB2 Optimizer for computing the amount of time and resources that a query will take to complete execution. The timeron is a combination of time, CPU utilization, disk I/O, and a few other factors. Due to the changing values of these parameters, the number of timerons needed to execute a query is dynamic and can change from execution to execution.

The timeron is also an invented unit of measurement; therefore, there is no a formula that can be used to translate the number of timerons it will take to execute a query into a time in seconds. That aside, timerons can help you determine if one query execution path is faster than another. (Don't worry if the number of timerons it takes to execute a query varies by ten or twenty between compilations - this could easily be due to changes in CPU activity, disk activity, or database usage.)

SQL translation

Before any SQL statement can be executed against a database, it must first be prepared. During this process the SQL statement is reduced down to an algebraic statement that the DB2 Optimizer can then analyze. This algebraic statement is referred to as the *query graph model,* and is worked with throughout the optimization process. Figure 6 shows the stages of optimization and parsing an SQL query must go through before it can be executed.

Figure 6. The SQL translation process

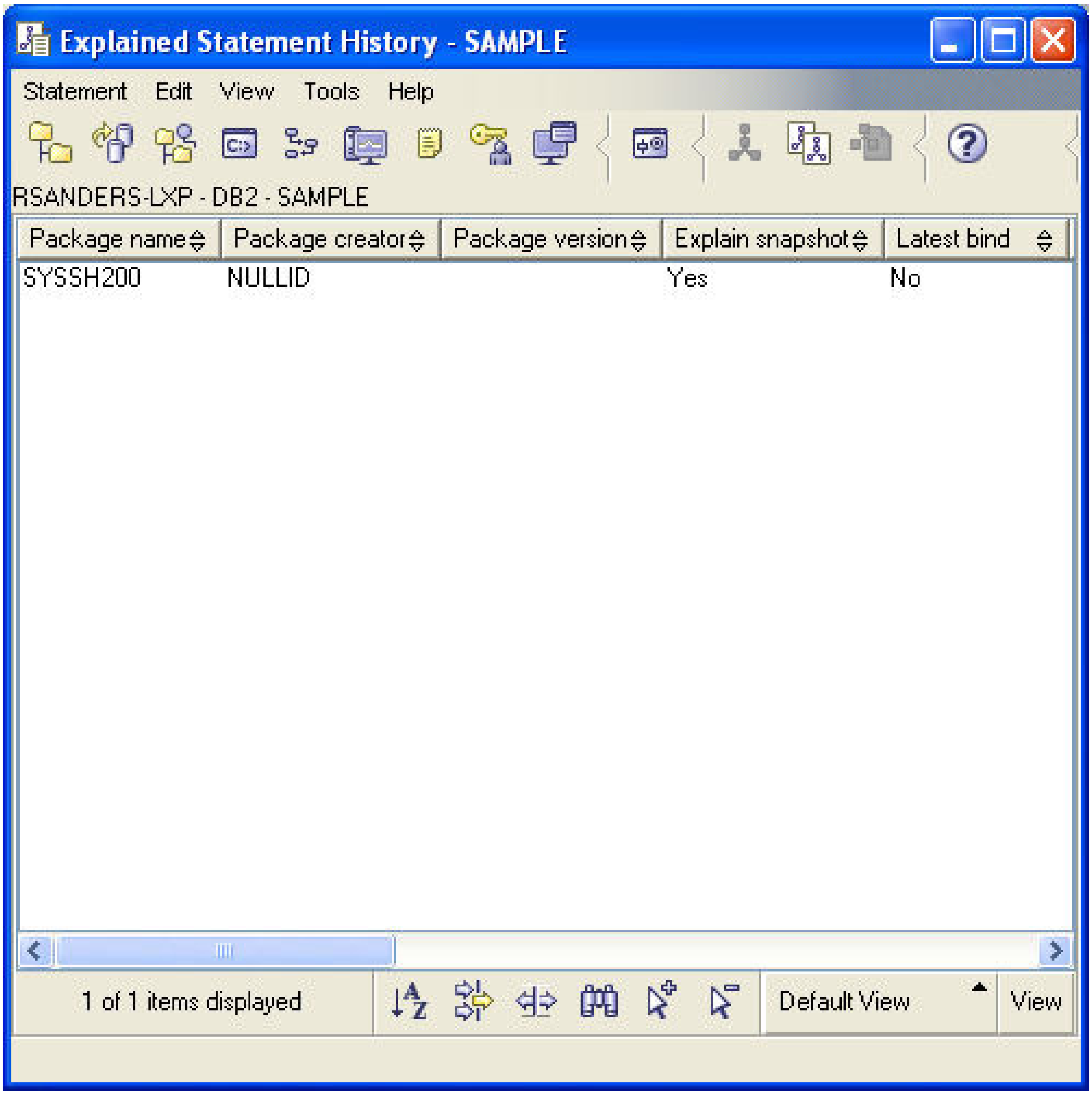


The final output of the optimization process is an *access plan.* The access plan is the path and steps that DB2 takes to execute the SQL statement. This is the information that is displayed by all of the explain tools available. At first, access plans appear to be quite complicated. But with practice, you soon discover that they are actually very easy to read and analyze.

Activating Visual Explain

Whenever comprehensive explain and/or explain snapshot data is collected, information about how and when the data was collected is recorded in the EXPLAIN_INSTANCE explain table. This information can be viewed at any time via the Explained Statement History window. The Explained Statement History window is activated by highlighting the appropriate database in the Control Center and selecting **Selected > Show Explained Statement History** from the Control Center menu. Figure 7 shows how the Explained Statement History window might look when explain snapshot data has been collected for one SQL statements.

Figure 7. The Explained Statement History window

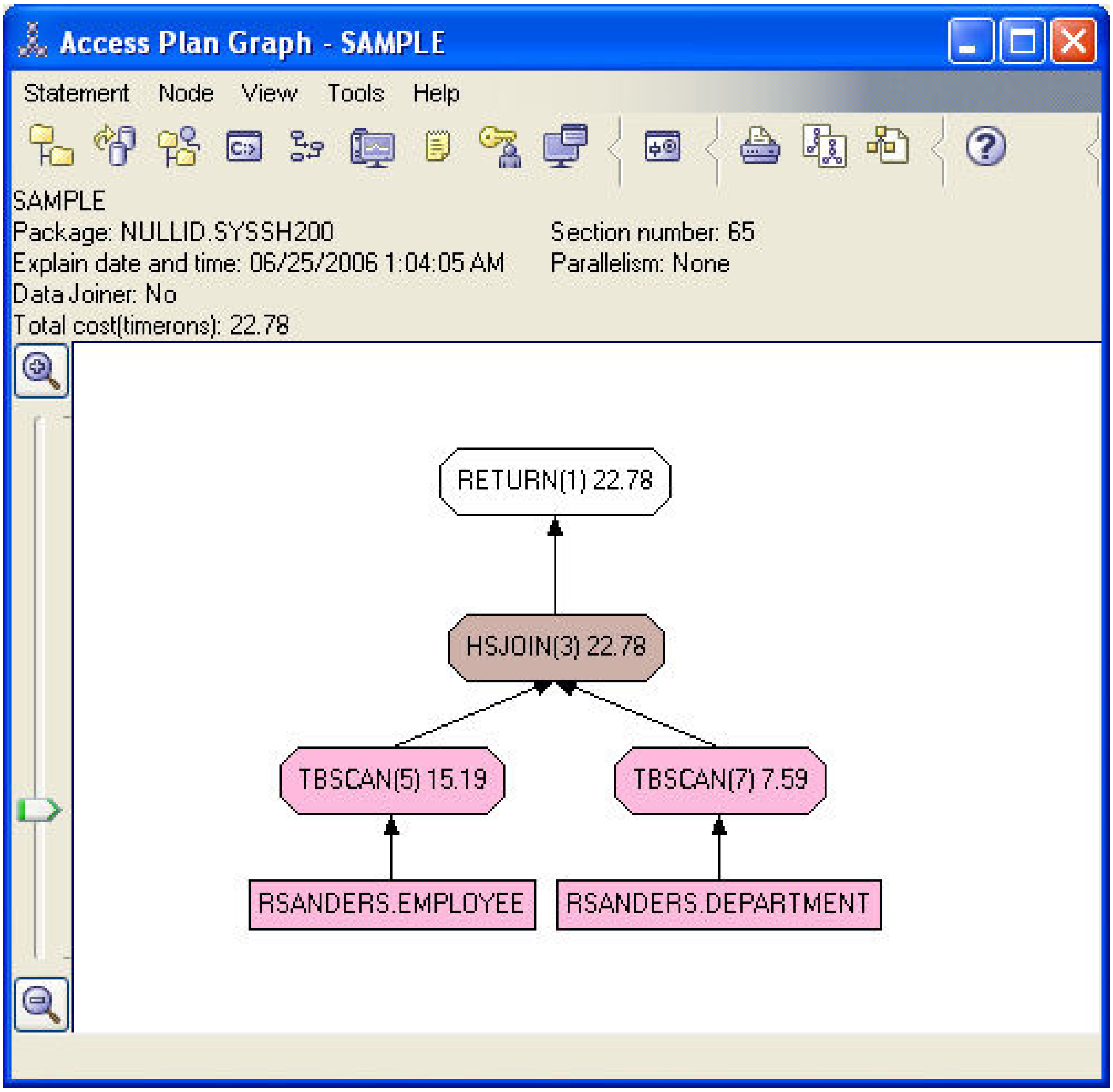


Package name	Package creator	Package version	Explain snapshot	Latest bind
SYSSH200	NULLID		Yes	No

Once the Explained Statement History window has been opened, Visual Explain can be used to analyze the explain snapshot data collected for any record shown by highlighting a record and selecting **Statement > Show Access Plan** from the Explained Statement History window's main menu. Figure 8 shows the Access Plan window that was created in this manner for the following query (which was ran against the SAMPLE database provided with DB2):

```
SELECT * FROM EMPLOYEE, DEPARTMENT WHERE WORKDEPT=DEPTNO
```

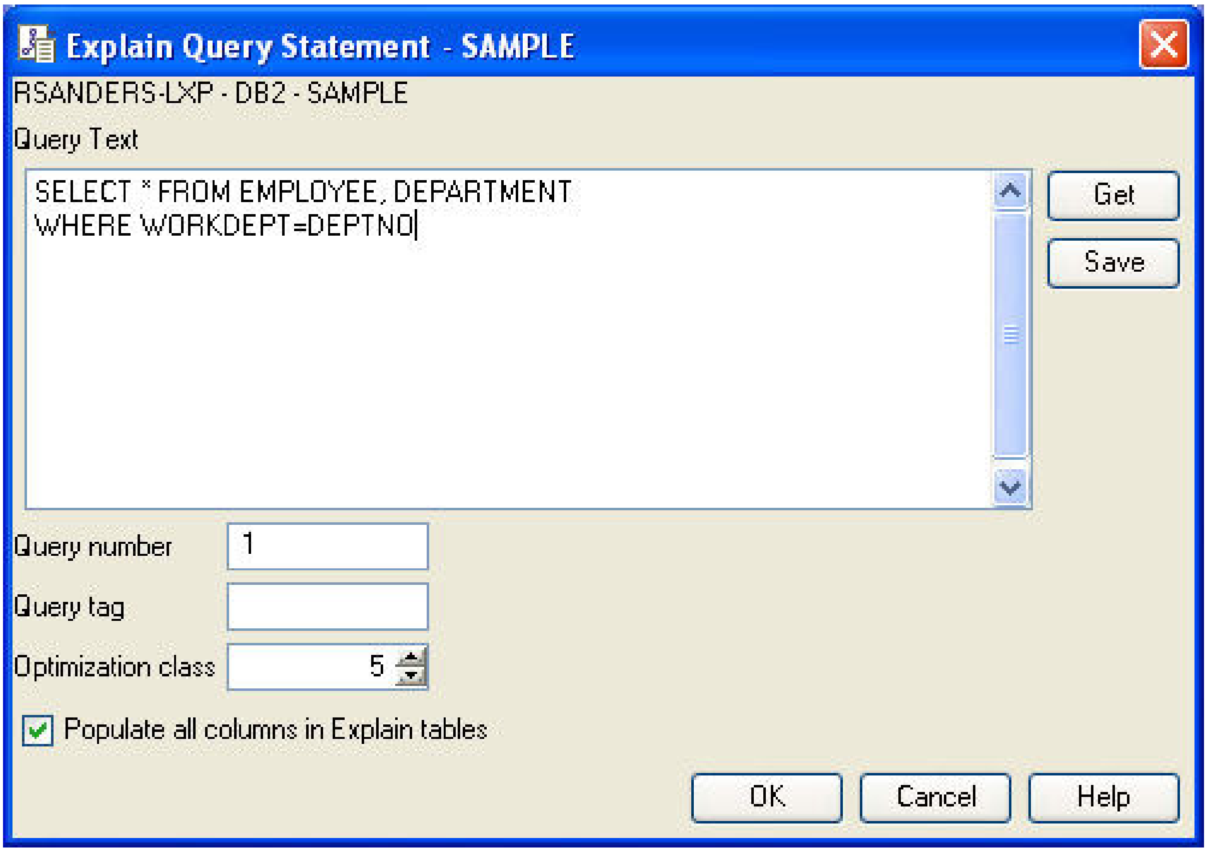
Figure 8. The Access Plan Graph window



SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 06/25/2006 1:04:05 AM
Parallelism: None
Data Joiner: No
Total cost(timerons): 22.78

Alternately, explain snapshot data can be collected for a new query and the corresponding access plan can be displayed by selecting **Statement > Explain Query...** from the Explained Statement History window's main menu. When these menu items are selected, the Explain Query Statement window opens and prompts you to enter the text for your query. Figure 9 shows what the Explain Query Statement window looks like after it has been populated with a simple query.

Figure 9. The Explain Query Statement window



Query Text

```
SELECT * FROM EMPLOYEE, DEPARTMENT
WHERE WORKDEPT=DEPTNO
```

Query number: 1

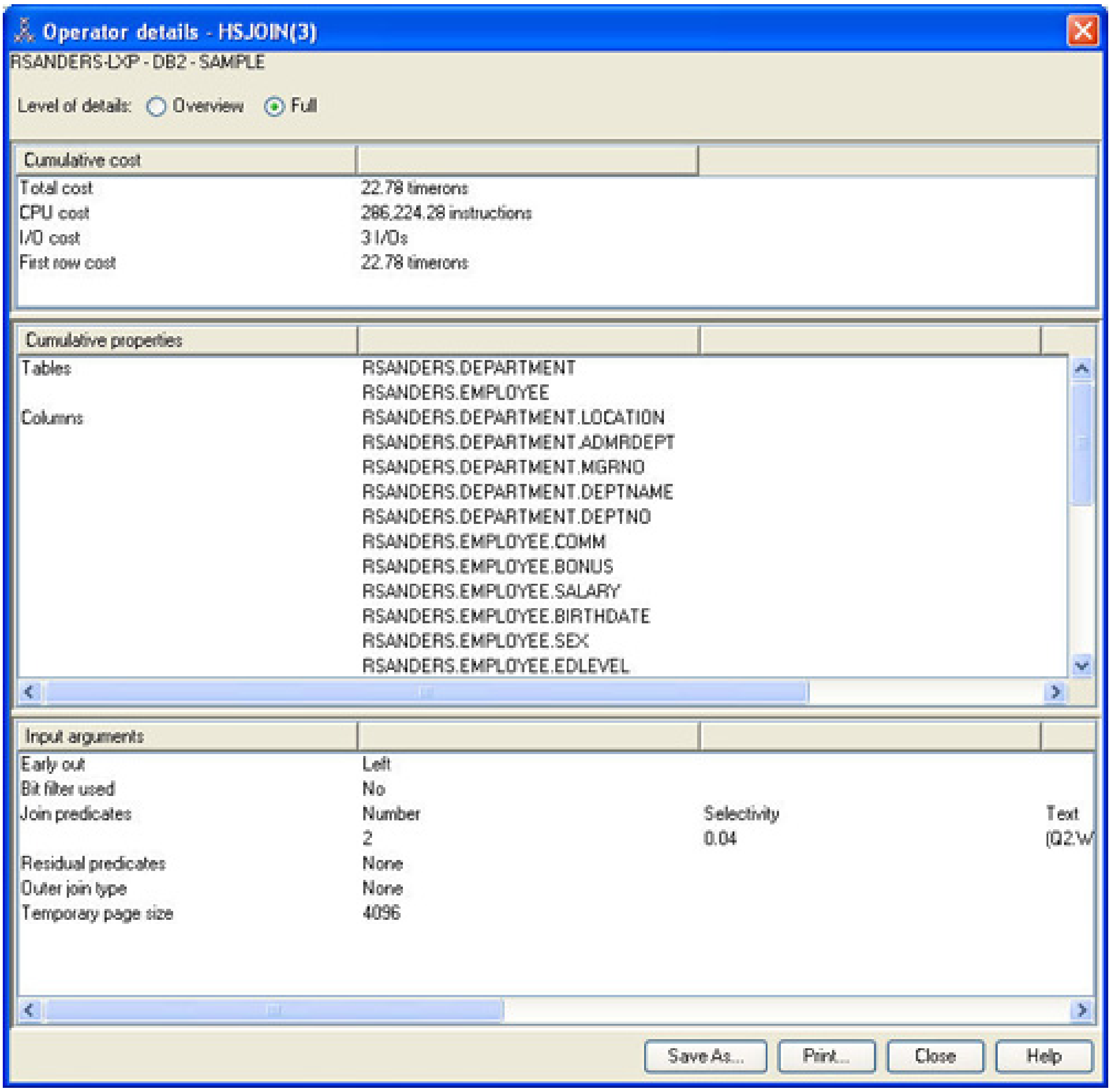
Query tag:

Optimization class: 5

☑ Populate all columns in Explain tables

Every component of the access plan shown in the Access Plan Graph window can be clicked to reveal more detailed information on that component. For example, if the HSJOIN(3) operator in the access plan shown in Figure 8 is selected, detailed information like that shown in Figure 10 might be displayed in the Operator details window.

Figure 10. The Operator details window



Operator details - HSJOIN(3)

RSANDERS-LXP - DB2 - SAMPLE

Level of details: ○ Overview ● Full

Cumulative cost		
Total cost	22.78 timerons	
CPU cost	286,224.28 instructions	
I/O cost	3 I/Os	
First row cost	22.78 timerons	

Cumulative properties			
Tables	RSANDERS.DEPARTMENT		
	RSANDERS.EMPLOYEE		
Columns	RSANDERS.DEPARTMENT.LOCATION		
	RSANDERS.DEPARTMENT.ADMRDEPT		
	RSANDERS.DEPARTMENT.MGRNO		
	RSANDERS.DEPARTMENT.DEPTNAME		
	RSANDERS.DEPARTMENT.DEPTNO		
	RSANDERS.EMPLOYEE.COMM		
	RSANDERS.EMPLOYEE.BONUS		
	RSANDERS.EMPLOYEE.SALARY		
	RSANDERS.EMPLOYEE.BIRTHDATE		
	RSANDERS.EMPLOYEE.SEX		
	RSANDERS.EMPLOYEE.EDLEVEL		

Input arguments			
Early out	Left		
Bit filter used	No		
Join predicates	Number	Selectivity	Text
	2	0.04	(Q2.W
Residual predicates	None		
Outer join type	None		
Temporary page size	4096		

When analyzing an access plan to locate performance bottlenecks, it is a good idea to try clicking through all the different object types to get comfortable with the query information that you have available.

Visual Explain components

You might have noticed that the output provided in the Access Plan window (refer to Figure 9) consists of a hierarchical graph that represents the various components that are needed to process the access plan that has been chosen for the query specified. Each component in the plan is represented as a graphical object known as a *node*. Two types of nodes can exist:

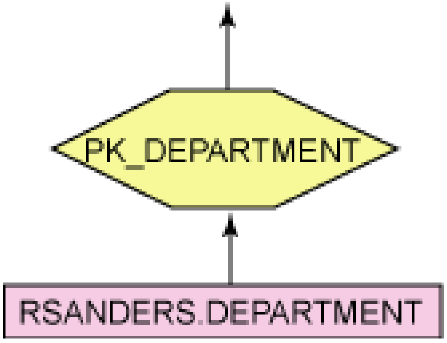
- **Operator**. An operator node is used to identify either an action that must be performed on data, or output produced from a table or index.

- **Operand**. An operand node is used to identify an entity on which an operation is performed (for example, a table would be the operand of a table scan operator).

Operands

Typically, operand nodes are used to identify tables, indexes, and table queues (table queues are used when intra-partition parallelism is used), which are symbolized in the hierarchical graph by rectangles (tables), diamonds (indexes), and parallelograms (table queues). Examples of table and index operands can be seen in Figure 11.

Figure 11. Table and index operands



Operators

Operator nodes, on the other hand, are used to identify anything from an insert operation to an index or table scan. Operator nodes, which are symbolized in the hierarchical graph by ovals, indicate how data is accessed, how tables are joined, and other factors such as whether or not a sort operation is to be performed. Table 6 lists the more common operators that can appear in an access plan hierarchical graph.

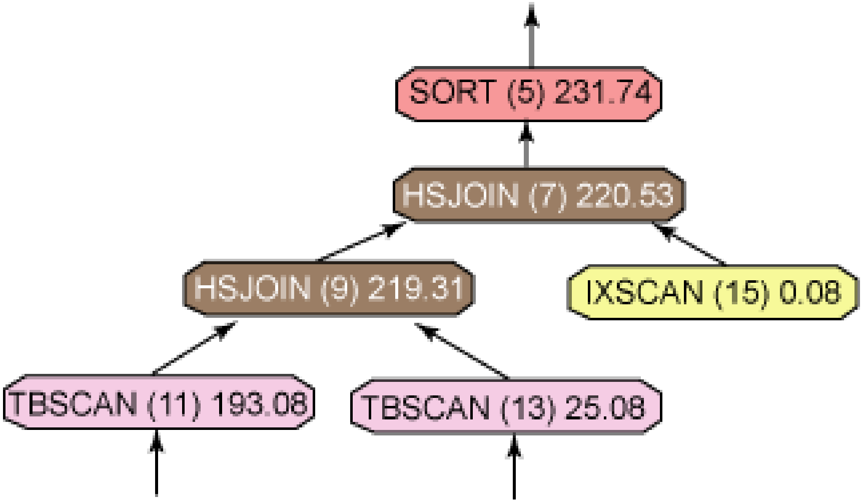
Table 6. Common Visual Explain operators

Operator	Operation performed
CMPEXP	Computes expressions. (For debug mode only.)
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.
PIPE	Transfers rows. (For debug mode only.)
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	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 data directly from the data pages.
TEMP	Stores data in a temporary table to be read back out (possibly multiple times).
TQUEUE	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.

Examples of some more common operands can be seen in Figure 12. In this example, three different actions are being performed: Two tables are having table scans performed, one index scan is being performed, and two data sets are being joined using the hashjoin algorithm.

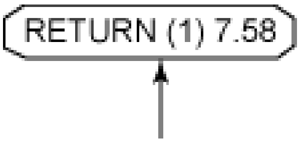
Figure 12. Several common operators



Connectors and the RETURN operator

Arrows that illustrate how data flows from one node to the next connect all nodes shown in the hierarchical graph and a RETURN operator normally terminates this path. The RETURN operator represents the final result set produced and contains summary information about the query and what is being returned from the completed SQL. The timeron value displayed with the RETURN the object represents the total length measurement of the time, in timerons, that was needed to complete the query. An example RETURN operator can be seen in Figure 13.

Figure 13. The RETURN operator



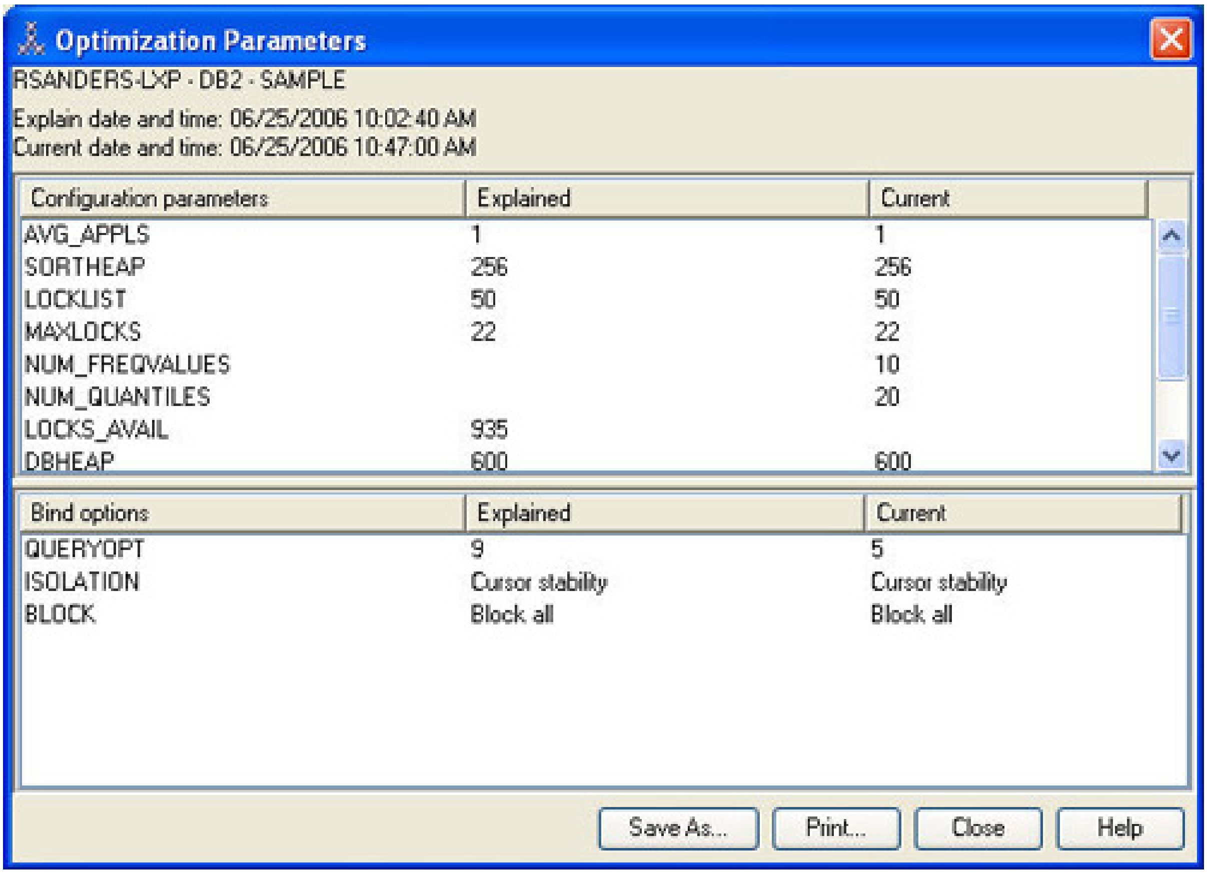
Factors that influence query performance

How a database environment has been configured and the query optimization level used to prepare a query can have a tremendous impact on how a query will be prepared, as well as how it will be executed.

Configuration parameter values

Visual Explain can quickly sumarize all of the parameters that affect query compilation and display them in a summary window. This window is called the Optimization Parameters window and it is invoked by selecting **Statement > Show Optimization Parameters** from the Access Plan Graph window's main menu. Figure 14 shows what the Optimization Parameters window might look like when it is activated.

Figure 14. The Optimization Parameters window



RSANDERS-LXP - DB2 - SAMPLE

Explain date and time: 06/25/2006 10:02:40 AM
Current date and time: 06/25/2006 10:47:00 AM

Configuration parameters	Explained	Current
AVG_APPLS	1	1
SORTHEAP	256	256
LOCKLIST	50	50
MAXLOCKS	22	22
NUM_FREQVALUES		10
NUM_QUANTILES		20
LOCKS_AVAIL	935	
DBHEAP	600	600

Bind options	Explained	Current
QUERYOPT	9	5
ISOLATION	Cursor stability	Cursor stability
BLOCK	Block all	Block all

Some of the configuration parameters shown on the Optimization Parameters window include:

- **AVG_APPLS** (average applications): This parameter indicates the average number of applications that will be running concurrently against the database. DB2 uses this information to determine how heavily the sort space and buffer pools will be used and how much space the query will likely be able to use.
- **SORTHEAP** (sort heap): The sort heap is the amount of space available in memory to perform a sort. If the sort requires more memory than is available for a sort heap, then part of the sort data will have to be paged to disk, (which can have a very negative impact on performance).
- **LOCKLIST** (lock list): This indicates the amount of memory available for DB2 to store locking information for each application. If the lock list space is quite small, then DB2 may have to escalate some locks to allow room for all the locks being held by the applications.
- **MAXLOCKS** (maximum lock list percentage): This parameter controls what percentage of the total lock list space one application can have. If an application tries to use up too much memory by having too many open locks, DB2 will escalate some of the locks to free up space in the lock list.
- **NUM_FREQVALUES** (number of frequency values): The number of frequency values is used by the DB2 Runstats utility to control how many

of the most frequent values DB2 will keep in memory. This information is used by the optimizer to determine what percentage of the total result set a predicate in a `WHERE` clause will eliminate.

- **NUM_QUANTILES** (number of data quantiles): The number of quantiles is used by the DB2 Runstats utility to control how many quantiles are captured for column data. Increasing the number of quantiles will give DB2 more information on the distribution of data in the database.
- **DBHEAP** (database heap): The database heap controls the amount of memory available for database object information. The objects include indexes, tables, buffer pools, and table spaces. Event monitor and log buffer information is stored here as well.
- **CPUSPEED** (CPU speed): The CPU speed of the computer. If the value is set to -1, then a CPU speed measurement program is used by DB2 to determine the proper setting.
- **BUFFPAGE and buffer pool size:** The optimizer uses the size of the available buffer pools in its optimization data. Increasing or decreasing the buffer pool size can have a significant impact on the access plan.

Optimization level used

The most important factor that can affect how an access plan is generated for a query is the *optimization level* that is used to prepare it. This information tells the DB2 Optimizer how much effort and what techniques should be used to determine the best access plan to use to resolve the query. A higher level will cause the optimizer to use more complex algorithms and algebraic analysis -- and as a result, will take much more time -- to generate the final plan.

Seven optimization classes are available and each class uses a different subset of all the rules and statistics available.The optimization classes available are:

- **0** -- Use a minimal amount of optimization
- **1** -- Use a degree of optimization roughly comparable to DB2/6000 Version 1, plus some additional low-cost features not found in Version 1
- **2** -- Use features of optimization class 5, but with a simplified join algorithm
- **3** -- Perform a moderate amount of optimization; similar to the query optimization characteristics of DB2 for MVS/ESA
- **5** -- Use a significant amount of optimization, with Heuristic Rules (Unless otherwise specified, this is the default optimization class used.)
- **7** -- Use a significant amount of optimization, without Heuristic Rules
- **9** -- Use all available optimization techniques

The following guidelines can be helpful when deciding on the best optimization class to use:

- Use optimization class **0** or **1** for queries that require very little optimization and rely heavily on primary key index searches or very simple joins (for example, very simple OLTP).
- Use optimization class **1** for simple queries that involve a small number of tables and joins involved indexes on the tables (for example, OLTP).
- Use optimization class **5** for a workload that involves complex OLTP or reports involving many complex joins on multiple tables (for example, mixed OLTP and reporting).
- Use optimization class **9** for queries that require significant analysis of data statistics and can run for a long time (over a minute) (for example, very complex data mining or decision support). The DB2 Optimizer will take much longer to produce an access plan but the improvements that can be found in the access plan normally outweigh the extra time needed to produce it.

Final thoughts on troubleshooting SQL

Entire books have been written on how to improve SQL performance so it's impossible to cover everything about query performance tuning here. However, here are a few key points that you should keep in mind when you begin using Visual Explain to trouble shoot a poorly performing query:

Lack of use of indexes. Is the query using the indexes you expect? Make sure that table scans are not occurring on tables you thought had indexes on them. This question can easily be answered by looking at the access plan diagram for the query. If the indexes do exist, then check the cardinality or the order of the index keys. It may not be what you expect.

Table cardinality and use of `'SELECT *'`. Sometimes the DB2 optimizer will decide that it is faster to scan an entire table due to the number of columns that you are bringing back. Perhaps the table is quite small, or perhaps it's just not efficient to scan an index and then return a large number of rows that return all the columns of the table. Try to return only the columns that you actually need. Take a look at what columns are being returned in each section of the query to see if you really need them and to see if that is why a table scan is occurring. Also, consider using include columns in an index.

Optimization level set too low. Many DBAs lower the optimization level to 1 to reduce the amount of time required for query preparation. Sometimes, raising the optimization level to 5 will allow the optimizer to find a better access plan without you having to create a new index to improve performance. This value can easily be adjusted in Visual Explain tool when you elect to generate explain information for a query using the Explain Query Statement window (see Figure 9). It can also be set from the Command Line processor by executing the following command:

```
SET CURRENT QUERY OPTIMIZATION [0|1|2|3|5|7|9]
```

Section 7. Other problem determination tools

db2mtrk utility

You have seen tools that can be used to examine the state of a database at a specific point in time (the snapshot monitor), collect data whenever a specific event or transition occurs (event monitors), and examine data access plans produced in response to queries (explain). There are two more tools that you need to be aware of if you are trying to locate a problem in your database environment. The first of these tools is known as the `db2mtrk` utility.

The `db2mtrk` utility is designed to provide a complete report of the memory status for instances, databases and agents. When executed, the `db2mtrk` command produces the following information about memory pool allocation:

- Current size
- Maximum size (hard limit)
- Largest size (high water mark)
- Type (identifier indicating function for which the memory pool will be used)
- Agent who allocated the pool (if the memory pool is private)

The `db2mtrk` utility is invoked by executing the `db2mtrk` command. The basic syntax for this command is:

```
db2mtrk
<-i>
<-d>
<-p>
<-m | -w>
<-r [Interval] <[Count]> >
<-v>
<-h>
```

where *Interval* identifies the number of seconds to wait between subsequent calls to the DB2 memory tracker and *Count* identifies the number of times to repeat calling the memory tracker.

How the `db2mtrk` utility collects and presents information is determined by the options specified when the `db2mtrk` command is invoked. Table 7 lists every option available and describes the behavior of each.

Table 7. db2mtrk command options available

Option	Meaning
-i	Show instance level memory
-d	Show database level memory
-p	Show private memory
-m	Show maximum values for each memory pool
-w	Show high water mark values for each memory pool
-r	Repeat mode
-v	Verbose output
-h	Display help information

If you want to obtain instance-level, database-level, and private memory pool allocation information, execute a `db2mtrk` command that looks like this:

```
db2mtrk -i -d -p
```

And when this command is executed, you might see something similar to the output shown below.

Sample output produced by the db2mtrk utility

```
Memory for instance

   monh      other
   320.0K    8.1M

Memory for database: SAMPLE

   utilh     pckcacheh catcacheh bph (1)   bph (S32K) bph (S16K) bph (S8K)
   64.0K     128.0K    64.0K     1.2M      704.0K     448.0K     320.0K

   bph (S4K) shsorth   lockh     dbh       other
   256.0K    0         320.0K    4.3M      128.0K

Memory for agent 3632

   other     apph      appctlh
   64.0K     64.0K     64.0K

Memory for agent 3184

   other     apph      appctlh
   64.0K     64.0K     64.0K

Memory for agent 508

   other     apph      appctlh
   448.0K    64.0K     64.0K
```

The db2pd utility

The `db2pd` utility is designed to retrieve information from appropriate DB2 database system memory sets and produce a thorough report that can be used to monitor and/or troubleshoot a database system (or any component of a database system). The db2pd utility is invoked by executing the db2pd command. The simplest form of this command is:

```
db2pd
<-inst>
<-database [DatabaseName] ,... | -alldatabases>
<-everything>
<-full>
```

where *DatabaseName* is the name assigned to one or more databases that information is to be obtained for.

If the db2pd command is executed with the -everything option specified, the db2pd utility will collect information for all elements of all databases on all database partition servers that are local to the server. If the -full option is used, the information produced will be expanded to its maximum length. (If this option is not specified, the information produced is truncated to save space on the display.)

In addition to collecting a large set of information for a database system, you can tell the db2pd utility to focus its collection on one specific area by specifying any of the following filter options as part of the db2pd command executed:

- -applications
- -fmp
- -agents
- -transactions
- -bufferpools
- -logs
- -locks
- -tablespaces
- -dynamic
- -static
- -fcm
- -memsets
- -mempools
- -memblocks
- -dbmcfg
- -dbcfg
- -catalogcache

- `-sysplex`
- `-tcbstats`
- `-reorg`
- `-recovery`
- `-reopt`
- `-osinfo`
- `-storagepaths`
- `-pages`

For example, if you wanted to obtain information about the transaction log files associated with the SAMPLE database, you could do so by executing a `db2pd` command that looks like this:

```
db2pd -database SAMPLE -logs
```

And when this command is executed, you might see something similar to the output shown below.

Sample output produced by the db2pd utility

```
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:43:48

Logs:
Current Log Number            0
Pages Written                 0
Method 1 Archive Status       n/a
Method 1 Next Log to Archive  n/a
Method 1 First Failure        n/a
Method 2 Archive Status       n/a
Method 2 Next Log to Archive  n/a
Method 2 First Failure        n/a

Address    StartLSN       State      Size       Pages      Filename
0x04BBD254 0x0000036B0000 0x00000000 1000       1000       S0000000.LOG
0x04BBD2F4 0x000003A98000 0x00000000 1000       1000       S0000001.LOG
0x04BBD394 0x000003E80000 0x00000000 1000       1000       S0000002.LOG
```

In some cases, when one of these filtering options is used, it in turn has its own set of options. Refer to the *DB2 Command Reference* for the complete syntax for the `db2pd` command.

Section 8. Summary

This tutorial was designed to introduce you to the set of monitoring tools that are available with DB2 9 and to show you how each are used to monitor how well (or how poorly) your database system is operating. Database monitoring is a vital activity that, when performed on a regular basis, provides continuous feedback on the health of a database system. Because database monitoring is such an integral part of database administration, DB2 9 comes equipped with a monitoring utility known as the database system monitor and although the name "database system monitor" suggests a single monitoring tool, in reality the database system monitor is composed of two distinct tools that can be used to capture and return system monitor information: a *snapshot monitor* and one or more *event monitors*.

The snapshot monitor allows you to capture a picture of the state of a database at a specific point in time while event monitors capture and log data as specific database events occur. It is important to note that bcause monitoring adds additional processing overhead, the amount of time spent monitoring a system should be limited, and monitoring should always be performed with a specific purpose in mind.

With DB2 Version 9, snapshot monitor data can also be obtained by using a set of SQL routines to access data stored in special administrative views.

Event monitors provide a way to collect monitor data when events or activities occur that cannot be monitored using the snapshot monitor. Additionally, while the snapshot monitor exists as a background process that begins capturing monitor data once a connection to a database has been established, event monitors must be specifically created before they can be used.

Because event monitors are special database objects that must be created before they can be used, they can only collect monitor data for events or transitions that take place in the database for which they have been defined. Event monitors cannot be used to collect monitor data at the instance level. You can create event monitors directly from the Control Center (select Create Event Monitor from the Event Monitors menu) or by executing the `CREATE EVENT MONITOR` SQL statement.

The health monitor adds a *management by exception* capability to DB2 9 by alerting administrators to potential system health issues *before* they become problems that affect a system's performance. The health monitor uses several health indicators to evaluate specific aspects of instance and database performance. Each health indicator acts as a precise measurement that the health monitor examines continuously to gauge the health of a particular aspect of a specific class of database objects. In turn, health indicators measure a finite set of distinct object states or a continuous range of values to determine whether an particular object is "healthy" or "unhealthy". Health indicators have a set of predefined threshold values, and the health monitor constantly compares the state of the system against these thresholds - if it finds that a particular threshold limit has been exceeded or detects an abnormal state for a particular object, it automatically issues an alert through the specified reporting channels.

The explain facility allows you to capture and view detailed information about the access plan chosen for a particular SQL statement, as well as performance information that can be used to help identify poorly written statements or a weakness in database design. Specifically, explain data helps you understand how the DB2 Database Manager accesses tables and indexes to satisfy a query. explain data can also be used to evaluate any performance tuning action taken. Before explain information can be captured, a special set of explain tables must be created.

Visual Explain is a GUI tool that provides database administrators and application developers with the ability to view a graphical representation of the access plan that has been chosen for a particular SQL statement.

However, Visual Explain can only be used to view explain snapshot data; to view explain data that has been collected and written to the explain tables, the `db2exfmt` tool must be used instead.

The `db2mtrk` utility is designed to provide a complete report of the memory status for instances, databases and agents.

The `db2pd` utility is designed to retrieve information from appropriate DB2 database system memory sets and produce a thorough report that can be used to monitor and/or troubleshoot a database system (or any component of a database system). The `db2pd` utility is invoked by executing the `db2pd` command.

Resources

Learn

- DB2 9 overview information. Find information about a new data server that includes patented pureXML® technology.
- Check out the other parts of the DB2 9 DBA exam 731 prep tutorial series.
- You can learn more about monitoring DB2 from the DB2 9 Information Center
- For more information on the DB2 9 for Linux, UNIX, and Windows Database Administration Certification (Exam 731), check out these links:
  - Information Management Training information
  - General certification information -- including some book suggestions, exam objectives, and courses.

Get products and technologies

- A trial version of DB2 9 is available for free download.
- Download DB2 Express-C, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.

Discuss

- Participate in the discussion forum for this content.

About the author

Roger E. Sanders
Roger E. Sanders is a Senior Manager - IBM Alliance Engineering at Network Appliance, Inc. He has been designing and developing databases and database applications for more than 20 years and has been working with DB2 Universal Database since it was first introduced with OS/2 1.3 Extended Edition. He has written articles for IDUG Solutions Journal, Certification Magazine, and developerWorks, presented and taught classes at IDUG and RUG conferences, participated in the development of the DB2 certification exams, writes a regular column for DB2 Magazine and is the author of 9 books on DB2 UDB.