This tutorial introduces the set of monitoring tools available with IBM DB2® 10.1 and shows how each is used to monitor how well (or how poorly) your database system is operating. This is fourth in a series of eight tutorials you can use to prepare for Part 4 of exam 611.

View more content in this series

Before you start

About this series

If you are preparing to take the DB2 10.1 for Linux®, UNIX®, and Windows® DBA certification exam (exam 611), you've come to the right place — a study hall of sorts. This series of DB2 certification preparation tutorials covers the major concepts you'll need to know for the test.

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 available with DB2 10.1 and to show you how each is used to monitor how well your database system is operating. In this tutorial, you will learn:

- How monitoring table functions work
- How administrative views offer an alternative interface for collected data
- How event monitors are created and how event monitor data is collected
- How monitoring tools operate with the workload manager
- How SQL (or XQuery) statement access plans are analyzed using the Explain facility
- How Visual Explain is used to view collected explain data
This is fourth in a series of eight tutorials to help you prepare for the DB2 10.1 for Linux, UNIX, and Windows DBA certification exam (exam 611). It is based on Part 4 of the previous edition of this series: "DB2 9 DBA certification exam 731 prep, Part 4: Monitoring DB2 activity." The material in this tutorial primarily covers the objectives in Section 4 of exam 611.

Objectives

After completing this tutorial, you should be able to:

- Use monitor table functions and administrative views to collect and view monitor data
- Create and activate event monitors
- Monitor workload manager features
- Capture and analyze Visual Explain information
- Identify the functions of the DB2 problem determination tools

Prerequisites

To take exam 611, you must have already passed the DB2 10.1 Fundamentals exam (exam 610). We recommend you study the DB2 10.1 fundamentals certification exam 610 prep series before starting this series. To help you understand some of the material presented here, 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 Workload Manager (WLM)**—A set of DB2 features that can help obtain a stable, predictable execution environment that meets business objectives. DB2 WLM manages system resources and governs application requests.
- **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 it if you download the free trial version of IBM DB2 10.1 to work along with this tutorial.

Monitoring table functions and administrative views

Database monitoring

Database monitoring is a vital activity that provides continuous feedback on the health of a database system. To facilitate monitoring, DB2 collects information from the database manager, its databases, and any connected applications. This collected information can be examined using one or more of the various monitoring interfaces provided with DB2.

There are two distinct methods for monitoring operations in a database. The first involves monitoring table functions and administrative views, and allows you to view information that shows
the state of various aspects of the database at a specific point in time. Event monitors, on the other hand, capture and log historical information as specific types of database events occur.

**Monitor elements**

Monitor elements are data structures that store information about a particular aspect of database system status. Data for monitored elements is continually accumulated in memory and available for querying by one or more of the various monitoring interfaces provided with DB2, such as table functions or event monitors. Each monitor element reflects one of the following types of data:

- **Counter**— A counter keeps a total count of the number of times an activity or event has occurred, increasing throughout the life of the monitor. Monitoring element counters are never reset, starting with database activation and continuing to accumulate until the database is deactivated. An example of a counter element would be the total number of SQL statements that have been executed against a database.

- **Gauge**— A gauge keeps 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.

- **Watermark**— A watermark indicates the highest (maximum) value an item has reached since monitoring began. An example of a watermark element would be the longest-running unit of work since the database was activated.

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

- **Timestamp**— A timestamp monitor element indicates the time an activity or event took place. Timestamp values are provided as the number of seconds and microseconds that have elapsed since 1 Jan 1970. An example of a timestamp element would be the date and time the first connection to a database was established. Compared to gauges that measure elapsed time (such as the time spent performing a sort), timestamps measure the exact point in time that something begins or ends.

It is important to note that monitoring elements are not exclusive to particular monitoring interfaces. Different table functions and event monitors collect monitoring data from sets of monitoring elements that can generally overlap. Monitor elements are categorized into three main categories:

- **Request monitor elements**— Request monitor elements measure the work carried out by the database server as a whole to process different types of requests. This includes overall database system processing and processing between server and connected clients. Examples of request monitor elements include the number of completed requests by the system, the total CPU usage time, and time spent waiting for the next request from an open connection.

- **Activity monitor elements**— Activity monitor elements are a subset of request monitor elements. These elements measure data server processing related to executing activities, in particular, work involved with SQL statement execution such as locking, sorting and row processing.
- **Data object monitor elements**— Data object monitor elements provide information about operations performed on particular data objects such as tables, indices, buffer pools, table spaces and containers.

DB2 offers hundreds of metrics across these three categories, but collecting the entire set can be counter-productive and misleading. Proper choice of relevant metrics for analysis leads the DBA to accurate deductions about system health or problems being investigated. Some examples of useful metrics are listed below:

- The ratio of database rows read to rows selected (ROWS_READ/ROWS_RETURNED) gives an indication of whether indices are being used properly to optimize reading operations.
- The time an agent waits for log records to be flushed to disk (LOG_WRITE_TIME/TOTAL_COMMITS) gives an indication of the efficiency of database transaction logging.

### Monitoring table functions

Monitoring table functions are designed to collect information about the state of the database manager, its databases, and any connected applications at a specific point in time. Monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure introduced in DB2 9.7. Prior to DB2 9.7, access to point in time monitoring data was achieved through snapshot monitoring, which uses a different monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product and usage of monitoring table functions is recommended. Monitoring table functions report information at the system, activity, and data object levels. These table functions collect data from request, activity and data object monitor elements, respectively.

### System monitoring table functions

The system monitoring perspective includes information about all work carried out by the database to process application requests. Request monitor elements continuously accumulate and aggregate this information from requests handled at various levels of the workload management (WLM) hierarchy. System monitoring information can be accessed at the unit of work, workload, service class, and connection:

- **MON_GET_UNIT_OF_WORK** and **MON_GET_UNIT_OF_WORK_DETAILS**— Return request metrics for one or more units of work
- **MON_GET_WORKLOAD** and **MON_GET_WORKLOAD_DETAILS**— Return request metrics for one or more workloads
- **MON_GET_SERVICE_SUBCLASS** and **MON_GET_SERVICE_SUBCLASS_DETAILS**— Return request metrics for one or more service subclasses
- **MON_GET_CONNECTION** and **MON_GET_CONNECTION_DETAILS**— Return request metrics for one or more connections

Table functions for monitoring system information are provided in pairs. One function allows relational access to commonly used data, having each request monitor data in one column. The
DETAILS table function returns an XML document encapsulating the full detailed set of available request monitor elements.

The database configuration parameter `mon_req_metrics` controls whether data is collected by request monitor elements across the entire data server, and can be set to the following values:

- **NONE**— No request monitor elements are collected.
- **BASE**— All request monitor elements are collected. This is the default value.
- **EXTENDED**— Similar to BASE, but additionally the values for some monitor elements are determined with more granularity.

Alternatively, collection can be controlled for individual WLM service classes through the `COLLECT REQUEST METRICS` clause while creating or altering service classes (`mon_req_metrics` and `COLLECT REQUEST METRICS` can be set to NONE, BASE, or EXTENDED).

If disabling of request metrics collection over a database named PAYROLL is desired, except for the service class ACCOUNTING, the following commands can be issued:

```sql
db2 update db cfg for payroll using mon_req_metrics none alter service class accounting collect request metrics extended
```

Accordingly, request metrics will only be collected for agents that run in the ACCOUNTING service class.

To identify the units of work that are consuming the highest amount of CPU time on the system, you could do so by executing the following command:

```sql
select application_handle, total_wait_time, total_rqst_time from table(mon_get_connection(null,null))
```

When this command is executed from the command-line processor, you should see something like the output shown below.

**Listing 1. Sample output from MON_GET_CONNECTION table function**

```plaintext
APPLICATION_HANDLE TOTAL_WAIT_TIME TOTAL_RQST_TIME
-------------------- ------------------- -------------------
--------------------- 39 179 269 78 0 0 51 207 316 77 0 21 50 1014 1408 40 109 351 79 89 167 7
record(s) selected.
```

The output produced by this command would help you decide if the percentage of time spent waiting for each application is acceptable. The SQL query can also be modified to produce a fourth column with the calculated percentage for time spent waiting.

**Activity monitoring table functions**

The activity monitoring perspective focuses on the subset of data server processing related to executing activities (SQL statements execution, in particular). Metrics are accumulated in memory and package cache for running activities and SQL statements. The following table functions access current data for activities:
• **MON_GET_ACTIVITY_DETAILS** returns data about individual activities in progress. Data is returned in a relational form, but detailed metrics are returned in an XML document in the DETAILS column of the result table.
• **MON_GET_PKG_CACHE_STMT** returns a current view of static and dynamic SQL statements in the database package cache. Data is returned in a relational form.
• **MON_GET_PKG_CACHE_STMT_DETAILS** returns detailed metrics for one or more package cache entries. Data is returned in a relational form, but the detailed metrics are returned in an XML document in the DETAILS column of the results table.

The database configuration parameter **mon_act_metrics** controls activity metrics collection in all workloads. Alternatively, collection can be controlled for individual WLM workloads through the **COLLECT ACTIVITY METRICS** clause while creating or altering workloads. Both can be set to **NONE**, **BASE**, or **EXTENDED**.

**Data object monitoring table functions**

The data object monitoring perspective provides information about operations carried out on database objects: tables, indices, buffer pools, table spaces, and containers. Each data object has monitoring elements incremented each time a request involves processing that object. The following table functions access current data for database objects:

• **MON_GET_BUFFERPOOL** returns monitor metrics for one or more buffer pools, allowing analysis of buffer pool activity, efficiency, hit ratio, etc.
• **MON_GET_TABLESPACE** returns monitor metrics for one or more table spaces, allowing analysis of table space and associated buffer pool activities.
• **MON_GET_CONTAINER** returns monitor metrics for one or more table space containers, allowing analysis of reading and writing activities.
• **MON_GET_TABLE** returns monitor metrics for one or more tables, including information about reads, inserts, updates, overflow activity, etc.
• **MON_GET_INDEX** returns monitor metrics for one or more indices, including information about the number of index scans, number of index-only scans, etc.

The database configuration parameter **mon_obj_metrics** controls data object metrics on the entire database for monitoring elements associated with **MON_GET_BUFFERPOOL**, **MON_GET_TABLESPACE**, and **MON_GET_CONTAINER** (WLM service class and workload clauses do not apply). Unlike their peers, monitoring elements associated with **MON_GET_TABLE** and **MON_GET_INDEX** table functions always collect data. Collection for tables and indices cannot be disabled by database configuration settings.

To list aggregated reading activity on all tables accessed since the database was activated, and ordered by highest reads, the following query can be issued.

**Listing 2. Query to list the aggregated reading activity on all tables accessed**

```
SELECT varchar(tabschema,20) as tabschema, varchar(tabname,20) as tabname, sum(rows_read) as total_rows_read,
FROM TABLE(MON_GET_TABLE('', '', -2)) AS t GROUP BY tabschema, tabname ORDER BY total_rows_read DESC
```
Listing 3. Sample output from MON_GET_TABLE table function

<table>
<thead>
<tr>
<th>TABSCHEMA</th>
<th>TABNAME</th>
<th>TOTAL_ROWS_READ</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSIBM</td>
<td>SYSSISTO 113</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSWORKL 22</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSROUTI 13</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSSERVI 13</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSTHRES 6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSTABLE 3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSCONTE 2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSDBAUT 2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSEVENT 2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSPLAN 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSSURRO 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSVERSI 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYXMLST 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSAUDIT 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSROLEA 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSROLES 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSTASKS 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSSWORKA 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>SYSIBM SYSSXMLPA 0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>19 record(s) selected.</td>
<td></td>
</tr>
</tbody>
</table>

The table below provides a summary of table functions with corresponding database configuration parameters that control monitoring data collection.

Table 1. Monitoring table functions

<table>
<thead>
<tr>
<th>Table functions</th>
<th>Database configuration control</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_UNIT_OF_WORK</td>
<td>mon_req_metrics</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORKDETAILS</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_DETAILS</td>
<td></td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS</td>
<td></td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASSDETAILS</td>
<td></td>
</tr>
<tr>
<td>MON_GET_CONNECTION</td>
<td></td>
</tr>
<tr>
<td>MON_GET_CONNECTIONDETAILS</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS</td>
<td></td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT</td>
<td>mon_act_metrics</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMTDETAILS</td>
<td></td>
</tr>
<tr>
<td>MON_GET_BUFFERPOOL</td>
<td></td>
</tr>
<tr>
<td>MON_GET_TABLESPACE</td>
<td></td>
</tr>
<tr>
<td>MON_GET_CONTAINER</td>
<td></td>
</tr>
<tr>
<td>MON_GET_TABLE</td>
<td></td>
</tr>
<tr>
<td>MON_GET_INDEX</td>
<td>Not applicable (always collected)</td>
</tr>
</tbody>
</table>

Administrative views

Administrative views are similar to table functions; they return data in table format. But unlike table functions, they do not require any input parameters. Administrative views belong to the SYSIBMADM schema and usually start with MON (monitoring table functions belong to the SYSPROC schema and usually start with MON_GET). The following table lists administrative views based on the new monitoring infrastructure.

Table 2. Administrative views

<table>
<thead>
<tr>
<th>Administrative view</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_BP_UTILIZATION</td>
<td>Returns key monitoring metrics, including hit ratios and average read and write times, for all buffer pools and all database partitions in the currently connected database.</td>
</tr>
<tr>
<td>MON_CONNECTION_SUMMARY</td>
<td>Returns key metrics for all connections in the currently connected database. It is designed to help monitor the system in a high-level manner, showing incoming work per connection.</td>
</tr>
<tr>
<td>MON_CURRENT_SQL</td>
<td>Returns key metrics for all activities that were submitted on all members of the database and have not yet been completed, including a point-in-time view of currently executing SQL statements (both static and dynamic) in the currently connected database.</td>
</tr>
</tbody>
</table>
MON_CURRENT_UOW
Returns key metrics for all units of work submitted on all members of the database. It can identify long-running units of work and to help prevent performance problems.

MON_DB_SUMMARY
Returns key metrics aggregated over all service classes in the currently connected database. It is designed to help monitor the system in a high-level manner by providing a concise summary of the database.

MON_LOCKWAITS
Returns information about agents working on behalf of applications that are waiting to obtain locks in the currently connected database. It is a useful query for identifying locking problems.

MON_PKG_CACHE_SUMMARY
Returns key metrics for both static and dynamic SQL statements in the cache, providing a high-level summary of the database package cache.

MON_SERVICE_SUBCLASS_SUMMARY
Returns key metrics for all WLM service subclasses in the currently connected database. It is designed to help monitor the system in a high-level manner, showing work executed per service class.

MON_TBSP_UTILIZATION
Returns key monitoring metrics, including hit ratios and utilization percentage, for all table spaces and all database partitions in the currently connected database.

MON_WORKLOAD_SUMMARY
Returns key metrics for all WLM workloads in the currently connected database. It is designed to help monitor the system in a high-level manner, showing incoming work per workload.

NOTE: Some of the monitoring administrative views above are designed to replace snapshot administrative views that use the pre-DB2 9.7 monitoring infrastructure, such as MON_LOCKWAITS (replaces SNAPLOCKWAIT) and MON_TBSP_UTILIZATION (replaces TBSP_UTILIZATION). Like their new counterparts, snapshot administrative views belong to the SYSIBMADM schema. For more information about snapshot monitoring and snapshot administrative views, refer to Part 4 of the previous edition of this series: "DB2 9 DBA certification exam 731 prep, Part 4: Monitoring DB2 activity" or the DB2 10.1 for Linux, UNIX, and Windows Database Monitoring Guide and Reference.

To identify and retrieve information on all applications that executed units of work exceeding one minute, you can issue the following query, which employs the MON_CURRENT_UOW administrative view.

**Listing 4. Query to identify and retrieve information on all applications exceeding 1 minute**

```sql
SELECT APPLICATION_HANDLE AS APPL_HANDLE, UOW_ID, ELAPSED_TIME_SEC, TOTAL_ROWS_MODIFIED AS TOTAL_READ, TOTAL_ROWS_MODIFIED AS TOTAL_MODIFIED FROM MON_CURRENT_UOW WHERE ELAPSED_TIME_SEC > 60 ORDER BY ELAPSED_TIME_SEC DESC
```

**Listing 5. Sample output from MON_CURRENT_UOW administrative view**

<table>
<thead>
<tr>
<th>APPL_HANDLE</th>
<th>UOW_ID</th>
<th>ELAPSED_TIME_SEC</th>
<th>TOTAL_READ</th>
<th>TOTAL_MODIFIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>254</td>
<td>1</td>
<td>750</td>
<td>87460</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>194</td>
<td>108</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>82</td>
<td>34</td>
<td>3</td>
</tr>
</tbody>
</table>

record(s) selected.

**Event monitoring**

**Event monitors**

Monitoring table functions provide report 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 reported by table functions.
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 table monitoring functions or the snapshot monitor because, in all likelihood, the deadlock cycle will have been broken long before data is collected from monitoring elements. 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 using table functions and event monitors: Table functions make use of data collected by monitoring elements in lightweight background processes that commence with database activation. In contrast, event monitors must be specifically created before they can be used. Several 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 monitor types

<table>
<thead>
<tr>
<th>Event monitor name</th>
<th>When the event data is generated</th>
<th>Data collected</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKING</td>
<td>Upon detection of lock timeouts or deadlocks or lock waits beyond a specified duration.</td>
<td>Consolidated comprehensive information regarding applications involved in any locking-related event, including the identification of participating statements, statement text, and a list of locks being held.</td>
</tr>
<tr>
<td>ACTIVITIES</td>
<td>Upon completion of an activity that executed in a WLM service class, workload or work class with COLLECT ACTIVITY DATA set.</td>
<td>Activity-level data for activities involving WLM objects.</td>
</tr>
<tr>
<td>STATEMENTS</td>
<td>End of SQL statement (for partitioned databases: end of SQL subsection).</td>
<td>Information about requests being made to the database with SQL statement execution, including start/stop time, statement text, fetch count, etc.</td>
</tr>
<tr>
<td>UNIT OF WORK</td>
<td>Upon completion of a unit of work.</td>
<td>Resource usage information and performance metrics for units of works that run on the system, including start/stop time, WLM information, etc. Recommended over the TRANSACTIONS event monitor.</td>
</tr>
<tr>
<td>PACKAGE CACHE</td>
<td>As entries are evicted from the package cache.</td>
<td>History of statements and related metrics for statements that are no longer in the package cache.</td>
</tr>
<tr>
<td>CONNECTIONS</td>
<td>End of connection.</td>
<td>Metrics and other monitor elements for each connection to the database by an application.</td>
</tr>
<tr>
<td>DATABASE</td>
<td>Database deactivation</td>
<td>Metrics and other monitor elements that reflect information about the database as a whole, including number of connections made to the database, rows of data inserted, etc.</td>
</tr>
<tr>
<td>BUFFERPOOLS</td>
<td>Database deactivation</td>
<td>Counters for buffer pools, prefetchers, page cleaners and direct I/O for each buffer pool.</td>
</tr>
<tr>
<td>TABLESPACES</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TABLES | Database deactivation | Table level counters, such as rows read or written, disk pages used by data, LOB or index objects.

STATISTICS | Collected automatically at regular intervals (database configuration parameter wlm_collect_int sets the interval). | Statistics computed from the activities that executed within each service class, workload, or work class that exists on the system.

THRESHOLD VIOLATIONS | Upon detection of a threshold violation. | Threshold violation information

CHANGE HISTORY | Upon monitor startup, when a configuration parameter or variable changes, or when a command, DDL, or utility completes. | Database and database manager configuration parameter changes, registry variable changes, execution of DDL statements, execution of certain DB2 utilities and commands, and change history event monitor startup.

Event monitors can report the data they collect in a number of ways. All event monitors can write the data they collect to tables; some write to unformatted event (UE) tables, which can help improve performance. Others can also write directly to a file or named pipe.

Creating event monitors

You create different types of event monitors by using variations on the CREATE EVENT MONITOR statement. You can use the options for that statement to specify the type of data that event monitors collect and how the event monitors produce their output. The basic variants of the CREATE EVENT MONITOR statement for each event monitor type are shown below:

**NOTE:** Parameters shown in angle brackets (&lt; &gt;) 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.

**Listing 6. CREATE EVENT MONITOR statement basic variants**

```
CREATE EVENT MONITOR [Name] FOR [DATABASE | TABLES | BUFFERPOOLS | TABLESPACES | STATISTICS | ACTIVITIES | THRESHOLD VIOLATIONS | CONNECTIONS &lt;WHERE [EventCondition]&gt; | STATEMENTS &lt;WHERE [EventCondition]&gt; , ...] WRITE TO [TABLE [GroupName] (TABLE [TableName]) | PIPE [PipeName] | FILE [PathName]] [MANUALSTART | AUTOSTART] CREATE EVENT MONITOR [Name] FOR [LOCKING | PACKAGE CACHE &lt;EventCondition&gt; | UNIT OF WORK] WRITE TO [TABLE [GroupName] (TABLE [TableName]) | UNFORMATTED EVENT TABLE [TableName]] [MANUALSTART | AUTOSTART] CREATE EVENT MONITOR [Name] FOR CHANGE HISTORY WHERE EVENT IN [EventControl , ...] WRITE TO [TABLE [GroupName] (TABLE [TableName])] [MANUALSTART | AUTOSTART]
```

where:

- **Name** identifies the name to be assigned to the event monitor being created.
- **EventCondition** identifies conditions used to determine which CONNECTION, STATEMENT, or SQL section within the PACKAGE CACHE the event monitor collects data for.
- **GroupName** identifies the logical data group for which the target table is defined (Choice depends on type of event monitor. Following logical data grouping of monitor elements, event monitors that write to tables generally produce one output table for each logical data group of monitor elements that they capture).
- **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.
• **PathName** 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 monitor data for both buffer pool and table space events and writes all data collected to a directory named `/export/home/bpts_data`. To do that, execute a `CREATE EVENT MONITOR` statement:

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

Now let's say you want to create an event monitor that captures all locking related monitoring data, including deadlocks, and have the data written to a UE table to minimize any potential effects on the performance of the database. To do that, execute a `CREATE EVENT MONITOR` statement:

```sql
CREATE EVENT MONITOR LOCKEVMON FOR LOCKING WRITE TO UNFORMATTED EVENT TABLE (TABLE LOCK_EVENTS)
```

Depending on the type of event monitor, output can be written to one or more database tables, one or more external files, a named pipe, or a UE tables. 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`, etc.). 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).

UE tables are relational tables, but they have only a limited number of columns. Most of the data associated with each event is written to a column containing an inline binary (BLOB) object. Writing event data in binary format reduces the time it takes to write each record to the table. For this reason, UE tables are particularly useful where event monitor performance is important.

### 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 `MANUALSTART`, 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` statement. The basic syntax for this statement is `SET EVENT MONITOR [MonitorName] STATE [0 | 1].`

To start an event monitor, you must specify the value 1. To stop an event monitor, specify the value 0.

Some event monitors will only collect data if their corresponding monitoring elements are enabled for collection (LOCKING, ACTIVITIES, STATISTICS, UNIT OF WORK, and PACKAGE CACHE). We have covered how to enable monitor elements collection using the database manager configuration and the WLM clause settings. Other event monitors collect data by default, such as TABLE, which starts collecting data automatically as soon as it is activated.
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:

```sql
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).

**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.

By default, records 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**

All event monitors can write their output to regular tables that can be queried directly using SQL. Some event monitors support other output options:

1. **Regular tables** — For a given event, each monitor element or metrics collected for the event is written to its own column in the table. This makes it possible to use a `SELECT` statement to query the output and examine the values for a specific monitor element.
2. **UE tables** — For a given event, most of the data is written to a column as an inline binary (BLOB) object. SQL cannot be used to extract legible data, post-processing on the UE table is required.
3. **Files** — Writing to files avoids any event monitor additional processing by the database manager and allows offline analysis of the extracted data. 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. The default setting for both parameters is NONE, which indicates that there is no space limit.

4. Pipes—Event monitor output can be written to a named pipe. This might be useful if you need to manipulate event data in real time by an external application.

Table 4 lists other supported output options for different types of event monitors.

Table 4. Output options for event monitors

<table>
<thead>
<tr>
<th>Event monitor</th>
<th>Regular table</th>
<th>UE table</th>
<th>File</th>
<th>Named pipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCKING</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTIVITIES</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>STATEMENTS</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT OF WORK</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACKAGE CACHE</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECTIONS</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATABASE</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>BUFFERPOOLS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>TABLESPACES</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>TABLES</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>STATISTICS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>THRESHOLD VIOLATIONS</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CHANGE HISTORY</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To access information collected in UE tables, a text report can be generated from the tables. Alternatively, data can be extracted into relational tables or XML so that it can be queried using SQL or pureXML. The following tools and methods are available:

- **db2evmonfmt tool**—A Java-based generic XML parser tool that produces a readable text output or a formatted XML output from UE tables. The tool allows selecting events of interest by filtering based on event ID, event type, time period, application, workload, or service class. The output format can also be completely controlled by creating your own XSLT style sheets instead of using the ones provided with db2evmonfmt.
- **EVMON_FORMAT_UETO_XML routine**—Extracts data from a UE table into an XML document.
- **EVMON_FORMAT_UETO_TABLES routine**—Extracts data from a UE table into a set of relational tables.

With the EVMON_FORMAT_UETO_XML and EVMON_FORMAT_UETO_TABLES routines, you can use a SELECT statement to specify the exact rows from the UE table that you want to extract.

To view event monitor data written to files or 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. To activate the event monitor productivity tool, execute the `db2evmon` command. The basic syntax for this command looks like `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.

**Guidelines for using event monitors**

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

Unlike monitoring table functions, most event monitors have a considerable impact on performance. This is due to the amount of information 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.

A common use for an event monitor is to capture deadlock information. 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 locking event monitor will collect information about 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 started much earlier by another application. Examine all the locks and applications involved to correctly determine where the problem originated.

Another common use for event monitors is to keep track of SQL statement processing. When an event monitor is used to capture information about every SQL statement executed, the properties of each statement, such as number of rows read, selected, deleted, etc., is recorded and is not presented as an aggregate total, as is the case with monitoring table functions. 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.
Workload management monitoring

The importance of WLM monitoring

A good workload management system helps to efficiently meet goals in the environment where work occurs. Workload management has three clearly defined domains:

1. Identification of the work entering the data server.
2. Management of the work when it is running.
3. Monitoring to ensure that the data server is being used efficiently.

The primary purpose of WLM monitoring is to validate the health and efficiency of your system and the individual workloads running on it. Using table functions, you can access real-time operational data, such as a list of running workload occurrences and the activities running in a service class or average response times. Using event monitors, you can capture detailed activity information and aggregate activity statistics for historical analysis.

WLM monitoring table functions

Looking at aggregate information reported by table functions should usually be the first step when you build a monitoring strategy. Aggregates give a good picture of overall data server activity and are also cheaper because you do not have to collect information on every activity in which you might be interested.

WLM monitoring table functions have names that begin with "WLM_" and can be used to obtain real-time monitoring data, such as information about work currently running on the system, statistics, and metrics for work performed on the system. This information can help you determine usage patterns and resource allocation and identify problem areas.

Compared to monitoring metrics table functions, WLM table functions generally provide computed value data that is more statistical in nature (such as averages, high watermarks, standard deviations, etc.) and provide a much more complex set of raw monitoring data.

Table 5 lists table functions used to monitor workload management features.

Table 5. WLM monitoring table functions

<table>
<thead>
<tr>
<th>Table function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES</td>
<td>Returns a list of workload occurrences assigned to a service class, including information about the current state, connection and activity.</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Returns a list of current activities associated with a workload occurrence, including information about the current state and type of the activity and the time at which the activity started.</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS</td>
<td>Returns a list of database agents associated with a service class or an application handle, including information about the state of the agent and the action the agent is performing.</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUPERCLASS_STATS</td>
<td>Shows summary statistics at the service superclass level — namely, high-water marks for concurrent connections to determine peak workload activity.</td>
</tr>
</tbody>
</table>
To list the occurrences of workloads mapped to the default user WLM service class SYSDEFAULTSUBCLASS, the following query uses the `WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES` table function to identify the source of activities.

**Listing 7. WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function**

```sql
SELECT APPLICATION_HANDLE, VARCHAR(WORKLOAD_NAME, 30) AS WORKLOAD, VARCHAR(SESSION_AUTH_ID, 16) AS SESSION_AUTH_ID, VARCHAR(APPLICATION_NAME, 10) AS APPL_NAME FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('SYSDEFAULTUSERCLASS', 'SYSDEFAULTSUBCLASS', -1)) AS T
```

**Listing 8. Sample output of WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES**

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>WORKLOAD</th>
<th>SESSION_AUTH_ID</th>
<th>APPL_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DB2INST1 db2bp</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

1 record(s) selected.

To show the current activities across database members that were created from a particular application's connection, the `WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES` table function can be used. You can use this information for a number of purposes, including identifying activities that might be causing problems on the data server. Using the sample output above, to learn about activities associated with application handle 28 for example, the following query can be issued.

**Listing 9. WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function**

```sql
SELECT APPLICATION_HANDLE, LOCAL_START_TIME, UOW_ID, ACTIVITY_ID, ACTIVITY_TYPE FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(28, -1)) AS T
```

**Listing 10. Sample output of WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES**

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>LOCAL_START_TIME</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>ACTIVITY_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2012-08-18-12.39.36.955048</td>
<td>25</td>
<td></td>
<td>READ_DML</td>
</tr>
</tbody>
</table>

1 record(s) selected.

The output shows that the activity associated with application handle 28 involves reading operations (`SELECT` statement). If more details about the activity are required, you can use the unit of work and activity IDs to dig down further using other WLM table monitoring functions.
WLM event monitors

DB2 workload management uses event monitors to capture information that might be of use in the future or for historical analysis. Within the set of event monitors covered above, three event monitors focus on WLM monitoring:

- **Activity event monitor**— The ACTIVITIES event monitor captures information about individual activities in a service class, workload, or work class, or activities that violated a threshold. Reported activity data can be used as input to tools, such as `db2advis` to acquire design recommendations that can enhance performance.

- **Threshold violation event monitor**— The THRESHOLD VIOLATIONS event monitor captures information when a threshold is violated. It indicates what threshold was violated, the activity that caused the violation, and what action was taken when it occurred.

- **Statistics event monitor**— The STATISTICS event monitor serves as a low-overhead alternative to capturing detailed activity information by collecting aggregate data, such as the number of activities completed, average execution time, etc. Aggregate data includes histograms for a number of activity measurements including lifetime, queue time, execution time, and estimated cost.

For example, to determine if there are outliers (large, expensive queries, possibly due to poor optimization of the queries themselves) in the database workload, the statistics event monitor can be used to report the average estimated cost of queries that run on the system, as well as the maximum estimated cost recorded. If the difference in cost is substantial, further investigation may be advisable. To create and enable the event monitor, issue a statement like this:

```
CREATE EVENT MONITOR DB2STATISTICS FOR STATISTICS WRITE TO TABLE SET EVENT MONITOR DB2STATISTICS STATE 1
```

To enable activity data collection for the default service class, issue the following statement:

```
ALTER SERVICE CLASS SYSDEFAULTSUBCLASS UNDER SYSDEFAULTUSERCLASS COLLECT AGGREGATE ACTIVITY DATA EXTENDED
```

Now with the event monitor activated and metrics collection enabled, the database workload should be left to run for some time so results obtained from the event monitor can be accurately indicative of the workload. At the end of this period, collection of statistics from memory into the statistics event monitor can be prompted by executing the `WLM_COLLECT_STATS` stored procedure:

```
CALL WLM_COLLECT_STATS();
```

`WLM_COLLECT_STATS` collects and reset statistics for WLM objects. All statistics tracked for service classes, workloads, threshold queues, and work action sets are sent to the active statistics event monitor (if one exists) and reset. If there is no active statistics event monitor, the statistics are only reset, but not collected.

Estimated cost statistics of queries are included with the different statistics written to the event monitor tables. To see them, query the service class statistics table `SCSTATS_DB2STATISTICS`. 
Listing 11. Querying the service class statistics table

SCSTATS_DB2STATISTICS

```sql
SELECT STATISTICS_TIMESTAMP, COORD_ACT_EST_COST_AVG, COST_ESTIMATE_TOP FROM SCSTATS_DB2STATISTICS WHERE SERVICE_SUPERCLASS_NAME = 'SYSDEFAULTUSERCLASS' AND SERVICE_SUBCLASS_NAME = 'SYSDEFAULTSUBCLASS'
```

Listing 12. Sample output from the SCSTATS_DB2STATISTICS table

```
STATISTICS_TIMESTAMP  COORD_ACT_EST_COST_AVG  COST_ESTIMATE_TOP
----------------------  ------------------------  ------------------------
2012-08-18-13.56.24.906302 26 101
```

The output shows that the maximum cost estimate is about four times the average cost, which indicates that queries running in this workload have comparable estimated costs. This is an indication that there wasn't significant outlier activity during that period.

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 indices 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 the statement is submitted from the Command Line Processor (CLP) or if the statement is a dynamic SQL statement in an application program (in other words, an SQL statement constructed at application runtime), the access plan is generated at the time the statement is issued and the executable form produced is stored temporarily in memory in the 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 package cache, the existing access plan is reused and the DB2 Optimizer is not invoked again.

Why is this important? Although the monitoring table functions and event monitors 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 to do that, you must use the 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 indices to satisfy a query. Explain data can also be used to evaluate any performance tuning action taken. In fact, anytime 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. The main explain tables, along with the information they are designed to hold, can be seen in Table 6.

Table 6. Explain tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN_ARGUMENT</td>
<td>Contains the unique characteristics for each individual operator used, if any.</td>
</tr>
<tr>
<td>EXPLAIN_INSTANCE</td>
<td>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.)</td>
</tr>
<tr>
<td>EXPLAIN_OBJECT</td>
<td>Contains information about the data objects required by the access plan generated for an SQL or XQuery statement.</td>
</tr>
<tr>
<td>EXPLAIN_OPERATOR</td>
<td>Contains all the operators needed by the SQL compiler to satisfy the SQL or XQuery statement.</td>
</tr>
<tr>
<td>EXPLAIN_PREDICATE</td>
<td>Contains information that identifies which predicates are applied by a specific operator.</td>
</tr>
<tr>
<td>EXPLAIN_STATEMENT</td>
<td>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 or enhanced with additional predicates by the SQL Precompiler). When an explain snapshot is requested, additional explain information is recorded to describe the access plan that was selected by the query optimizer. This information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT table.</td>
</tr>
<tr>
<td>EXPLAIN_STREAM</td>
<td>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.)</td>
</tr>
</tbody>
</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 before the Explain facility can be used. Fortunately, creating explain tables is quite straightforward. There are two methods:

1. Run the EXPLAIN.DDL script in the misc subdirectory of the sqllib subdirectory.
2. Use the SYSPROC.SYSINSTALLOBJECTS procedure. You can also use this procedure to drop and validate explain tables.

Collecting explain data

The Explain facility incorporates several individual tools, and not all tools require the same kind of explain data. Therefore, two 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.

There are a variety of ways in which both types of explain data can be collected, including:

• 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 BIND or PREP command.
• Using the EXPLSNAP bind option with the or BIND or PREP command.

**EXPLAIN SQL statement**

One way to collect 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] &lt;FOR SNAPSHOT | WITH SNAPSHOT&gt; FOR [SQLStatement]
```

*SQLStatement* identifies the SQL statement that explain data or explain snapshot data is to be collected for. (This statement can be any valid CALL, Compound SQL (Dynamic), DELETE, INSERT, MERGE, REFRESH, SELECT, SELECT INTO, SET INTEGRITY, UPDATE, VALUES, or VALUES INTO statement, or any valid XQuery 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, comprehensive explain information and 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 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`. To collect only explain snapshot data for the same SQL statement, execute an **EXPLAIN** statement like this: `EXPLAIN ALL FOR SNAPSHOT FOR SELECT * FROM DEPARTMENT`. And to collect only comprehensive explain data, execute an **EXPLAIN** statement 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.
CURRENT EXPLAIN MODE and the CURRENT EXPLAIN SNAPSHOT special registers

Although the EXPLAIN SQL statement is useful when you want to collect explain 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 snapshot information is collected for each statement processed. (The statements themselves, however, may or may not be executed once explain snapshot information has been collected.)

The two Explain facility special registers 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.

Listing 13. Basic syntax for the SET CURRENT EXPLAIN MODE SQL statement

```
SET CURRENT EXPLAIN MODE &lt;=\&gt; [NO | YES | EXPLAIN | REOPT | RECOMMEND INDEXES | EVALUATE INDEXES |
RECOMMEND PARTITIONINGS | EVALUATE PARTITIONINGS]
```

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

```
SET CURRENT EXPLAIN SNAPSHOT &lt;=\&gt; [YES | NO | EXPLAIN | REOPT]
```

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. But if either special register is set to EXPLAIN, the Explain facility is activated and comprehensive explain information or 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 snapshot data has been collected.

If either the CURRENT EXPLAIN MODE or the CURRENT EXPLAIN SNAPSHOT special register are set to REOPT, the Explain facility is activated, and explain information or snapshot data (or both if both special registers have been set) is captured whenever a static or dynamic SQL statement is processed during statement re-optimization at execution time (i.e., 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 snapshot data can be collected for dynamic SQL statements. But often, database applications are composed 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 or snapshot data for static and/or dynamic SQL statements coded in an embedded SQL application, you rely on the EXPLAIN and EXPLSNAP precompile/bind options.

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 explain snapshot data is collected. One or both of these options can be specified as part of the PRECOMPILE (or PREP) command 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 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 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 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 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 runtime, even if the CURRENT EXPLAIN MODE 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 information and snapshot data can be collected. But once the data is collected, how can it be viewed?

Several methods are available to give you the flexibility to capture, display, and analyze explain information, including:

- Writing your own queries against the explain tables
- db2expln tool (does not need captured explain data)
- db2exfmt tool
- Visual Explain

Writing your own queries against explain tables enables the easy manipulation of output, comparisons among different queries, or comparisons among executions of the same query over time.

**db2expln**

The db2expln tool describes the access plan selected for SQL and XQuery statements. Use the tool to obtain a quick explanation of the chosen access plan when explain data was not captured.
For static SQL and XQuery statements, db2expln examines the packages stored in the system catalog tables. For dynamic SQL and XQuery statements, db2expln examines the query cache sections. This utility shows the actual implementation of the chosen access plan; it does not show optimizer information. By examining the generated access plan, the db2expln tool provides a relatively compact, verbal overview of the operations that will occur at runtime.

**db2exfmt**

Unlike the db2expln tool, the db2exfmt tool is designed to work directly with comprehensive explain information or snapshot data 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 chosen for a particular SQL statement. In addition, Visual Explain allows you to:

- See the database statistics used to optimize the SQL statement.
- Determine whether 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 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 collected and written to the explain tables, the db2exfmt tool must be used instead.

As you can see, the various tools available for displaying comprehensive explain information and explain snapshot data vary greatly in their complexity and the capabilities they provide. Table 7 summarizes the 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 7. Comparison of available Explain facility tools**

<table>
<thead>
<tr>
<th>Desired characteristics</th>
<th>Explain tables</th>
<th>db2expln</th>
<th>db2exfmt</th>
<th>Visual Explain</th>
</tr>
</thead>
<tbody>
<tr>
<td>User interface</td>
<td>None</td>
<td>Text-based</td>
<td>Text-based</td>
<td>Graphical</td>
</tr>
<tr>
<td>“Quick and dirty” static SQL and XQuery analysis</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static SQL and XQuery support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic SQL and XQuery support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CLI application support</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

As you can see, the various tools available for displaying comprehensive explain information and explain snapshot data vary greatly in their complexity and the capabilities they provide. Table 7 summarizes the 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.
Visual Explain — A closer look

Timerons and SQL translation

Timerons

The most important thing to 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, so there is no 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 10 or 20 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 1 shows the stages of optimization and parsing an SQL query must go through before it can be executed.

Figure 1. The SQL translation process

---

<table>
<thead>
<tr>
<th>Detailed optimizer information</th>
<th>Yes</th>
<th></th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suited for analysis of multiple statements</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Information is accessible from within an application</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 displayed by all of the explain tools available.

Visual Explain with IBM Data Studio

IBM Data Studio is the recommended DB2 administration tool and replaces the deprecated Control Center tools. IBM Data Studio provides an integrated, modular Eclipse-based environment for database development and administration of DB2 for Linux, UNIX, and Windows. It also provides collaborative database development tools for DB2 for z/OS®, DB2 for System i®, Informix®, and other non-IBM databases. It is available at no charge (download IBM Data Studio). To activate Visual Explain, we will use the IBM Query Tuning perspective. Refer to the IBM Data Studio Version 3.1.1 information center for details on how to switch perspectives.

After connecting to the database in question, to visualize the access plan for a query, simply write the query in the script editor window and right-click anywhere on the query and choose Open Visual Explain. Figure 2 shows the access plan window created for the following query, which was run against the SAMPLE database provided with DB2): SELECT * FROM EMPLOYEE, DEPARTMENT WHERE WORKDEPT=DEPTNO.

Figure 2. Access plan diagram

On the left part of the access plan diagram, you can click on the Overview of Diagram tab, which displays general information. The Description of Selected Node tab displays information about the currently selected node in the diagram. Every component of the access plan can be clicked to reveal detailed information about that component. For example, if the HSJOIN(2) operator in the access plan shown in Figure 2 is selected, detailed information like that shown in Figure 3 might be displayed in the Description of Selected Node tab.
When analyzing an access plan to locate performance bottlenecks, it's a good idea to click through the different object types to get comfortable with the query information available.

**Visual Explain components**

The access plan consists of a hierarchical graph that represents the various components 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 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, indices, and table queues (table queues are used when intra-partition parallelism is used), symbolized in the hierarchical graph by rectangles (tables), diamonds (indexes), and parallelograms (table queues).

**Operators**

Operator nodes are used to identify anything from an insert operation to an index or table scan. Operator nodes, symbolized in the hierarchical graph by ovals, indicate how data is accessed, how tables are joined, and other factors such as whether a sort operation is to be performed. **Table 8** lists the more common operators that can appear in an access plan hierarchical graph.
Table 8. Common Visual Explain operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operations performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>Deletes rows from a table.</td>
</tr>
<tr>
<td>EISCAN</td>
<td>Scans a user-defined index to produce a reduced stream of rows.</td>
</tr>
<tr>
<td>FETCH</td>
<td>Fetches columns from a table using a specific record identifier.</td>
</tr>
<tr>
<td>FILTER</td>
<td>Filters data by applying one or more predicates to it.</td>
</tr>
<tr>
<td>GENROW</td>
<td>Generates a table of rows.</td>
</tr>
<tr>
<td>GRPBY</td>
<td>Groups rows by common values of designated columns or functions and evaluates set functions.</td>
</tr>
<tr>
<td>HSJOIN</td>
<td>Represents a hash join, where two or more tables are hashed on the join columns.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Inserts rows into a table.</td>
</tr>
<tr>
<td>IXAND</td>
<td>ANDs together the row identifiers (RIDs) from two or more index scans.</td>
</tr>
<tr>
<td>IXSCAN</td>
<td>Scans an index of a table with optional start/stop conditions, producing an ordered stream of rows.</td>
</tr>
<tr>
<td>MSJOIN</td>
<td>Represents a merge join, where outer and inner tables must be in join-predicate order.</td>
</tr>
<tr>
<td>NLJOIN</td>
<td>Represents a nested loop join that accesses an inner table once for each row of the outer table.</td>
</tr>
<tr>
<td>PIPE</td>
<td>Transfers rows. (For debug mode only.)</td>
</tr>
<tr>
<td>RETURN</td>
<td>Represents the return of data from the query to the user.</td>
</tr>
<tr>
<td>RIDSCN</td>
<td>Scans a list of row identifiers (RIDs) obtained from one or more indexes.</td>
</tr>
<tr>
<td>RPD</td>
<td>An operator for remote plans. It is similar to the SHIP operator in Version 8 (RQUERY operator in previous versions), except that it does not contain an SQL or XQuery statement.</td>
</tr>
<tr>
<td>SHIP</td>
<td>Retrieves data from a remote database source. Used in the federated system.</td>
</tr>
<tr>
<td>SORT</td>
<td>Sorts rows in the order of specified columns, and optionally eliminates duplicate entries.</td>
</tr>
<tr>
<td>TBSCAN</td>
<td>Retrieves rows by reading all data directly from the data pages.</td>
</tr>
<tr>
<td>TEMP</td>
<td>Stores data in a temporary table to be read back out (possibly multiple times).</td>
</tr>
<tr>
<td>TQUEUE</td>
<td>Transfers table data between database agents.</td>
</tr>
<tr>
<td>UNION</td>
<td>Concatenates streams of rows from multiple tables.</td>
</tr>
<tr>
<td>UNIQUE</td>
<td>Eliminates rows with duplicate values for specified columns.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Updates rows in a table.</td>
</tr>
<tr>
<td>XISCAN</td>
<td>Scans an index of an XML table.</td>
</tr>
<tr>
<td>XSCAN</td>
<td>Navigates an XML document node subtrees.</td>
</tr>
<tr>
<td>XANDOR</td>
<td>Allows ANDed and ORed predicates to be applied to multiple XML indexes.</td>
</tr>
</tbody>
</table>
Examples of some more common operands can be seen in Figure 4. In this example, three 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 4. 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 values accumulate from bottom to top, and the value displayed with the RETURN the object represents the total length measurement of the time, in timerons, needed to complete the query.

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

The access plan diagram also shows a summary of values for parameters that affect query compilation at the time the access plan was generated (under Environment). Figure 5 shows the configuration values for the above access plan diagram.
Figure 5. Configuration parameters

Configuration parameters that affect query compilation include:

- **AVG_APPLS**— 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**— 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 negative impact on performance.
• **LOCKLIST**— This indicates the amount of memory available for DB2 to store locking information for each application. If the lock list space is quite small, DB2 may have to escalate some locks to allow room for all the locks being held by the applications.

• **MAXLOCKS**— 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.

• **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, a CPU speed measurement program is used by DB2 to determine the proper setting.

• **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 classes

Optimization classes determine how the optimizer chooses the most efficient access plan for the running statement. Optimization classes differ in the number and type of optimization strategies considered during the compilation of a query. Although you can specify optimization techniques individually to improve runtime performance for the query, the more optimization techniques you specify, the more time and system resources query compilation will require.

You can specify one of the following optimization classes when you compile an SQL or XQuery statement:

- **0** — Use a minimal amount of optimization.
- **1** — Similar to class 0, except that merge scan joins and table scans are also available.
- **2** — Use features of optimization class 5, but with a simplified join algorithm.
- **3** — Perform a moderate amount of optimization, coming closest to matching the query optimization characteristics of DB2 for z/OS.
- **5** — Use a significant amount of optimization using all available statistics and query rewrite rules, reducing optimization when detecting that additional resources and processing time for complex queries are not warranted (unless otherwise specified, this is the default optimization class used).
- **7** — Similar to class 5, except that the optimizer never considers reducing the amount of query optimization for complex queries.
- **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, simple OLTP).
- Use optimization class **1** for simple queries that involve a small number of tables and joins involving indexes on the tables (OLTP, for example).
- 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 (more than 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 to keep in mind when you troubleshoot a poorly performing query.

Lack of use of indices — Is the query using the indices you expect? Make sure table scans are not occurring on tables you thought had indices on them. This question can easily be answered by looking at the access plan diagram for the query. If the indices do exist, 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 set from the command-line processor by executing the following command: SET CURRENT QUERY OPTIMIZATION [0|1|2|3|5|7|9].

Other system-based DB2 tools

The db2val and db2ls utilities

You have seen tools that can be used to examine the state of a database at a specific point in time (table monitoring functions and administrative views), collect data whenever a specific event or transition occurs (event monitors), and examine data access plans produced in response to queries (explain). There are other system-based tools that can be used to learn about the DB2 environment or to help locate a problem in your database environment.

db2val

db2val, or the DB2 copy validation tool, can verify the basic functions of a DB2 copy by checking the state of installation files, instance setup, and local database connections. The basic syntax for the db2val command is shown below. Table 9 lists available options and describes corresponding db2val behavior.
Listing 14. Basic syntax for db2val command

```
db2val &lt;-o&gt; &lt;-i [instance_name], ... &gt; &lt;-a&gt; &lt;-s&gt; &lt;-b [database_name]&gt; &lt;-l [log_file]&gt; &lt;-t [trace_file]&gt; &lt;-h&gt;
```

The `instance_name` specifies the instance to validate (defaults to current instance if not specified), `database_name` specifies the database to validate, `log_file` specifies the output log file for the command, and `trace_file` specifies the trace file path for Linux and UNIX systems.

### Table 9. db2val command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-o</code></td>
<td>Validate installation files only (does not validate instance, database and extended security)</td>
</tr>
<tr>
<td><code>-i</code></td>
<td>Validate the specified instance(s)</td>
</tr>
<tr>
<td><code>-a</code></td>
<td>Validate all instances in the DB2 copy</td>
</tr>
<tr>
<td><code>-s</code></td>
<td>Starts the DB2 database manager for the specified instance that is part of a partitioned database environment</td>
</tr>
<tr>
<td><code>-b</code></td>
<td>Validates database creation and connections to the database specified. Only active databases are verified</td>
</tr>
<tr>
<td><code>-l</code></td>
<td>Writes the log to the file name specified</td>
</tr>
<tr>
<td><code>-t</code></td>
<td>Specifies the full path and name of trace file (Linux and UNIX only)</td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Display usage information for the db2val command</td>
</tr>
</tbody>
</table>

**db2ls**

The `db2ls` command lists the DB2 products and features installed on your Linux and UNIX systems, including DB2 HTML documentation. With the ability to install multiple copies of DB2 products on your system and the flexibility to install DB2 products and features in the path of your choice, you can use the `db2ls` command to list where DB2 products are installed on your system and the DB2 product level.

**The db2mtrk utility**

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

The `db2mtrk` utility is invoked by executing the `db2mtrk` command.
Listing 15. Basic syntax for the db2mtrk command

db2mtrk &lt;-i&amp;gt; &amp;lt;-d&amp;gt; &amp;lt;-m | -w&amp;gt; &amp;lt;-a&amp;gt; &amp;lt;-r [Interval] &amp;lt;[Count]&amp;gt; &amp;gt; &amp;lt;-v&amp;gt; &amp;lt;-h&amp;gt;

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 10 lists options available and describes the behavior of each.

Table 10. db2mtrk command options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i</td>
<td>Show instance-level memory</td>
</tr>
<tr>
<td>-d</td>
<td>Show database-level memory</td>
</tr>
<tr>
<td>-m</td>
<td>Show maximum values for each pool</td>
</tr>
<tr>
<td>-w</td>
<td>Show high-watermark values for each pool</td>
</tr>
<tr>
<td>-a</td>
<td>Show application memory usage</td>
</tr>
<tr>
<td>-r</td>
<td>Repeat mode</td>
</tr>
<tr>
<td>-v</td>
<td>Verbose output</td>
</tr>
<tr>
<td>-h</td>
<td>Show help screen</td>
</tr>
</tbody>
</table>

To obtain instance-level, database-level, and application information, execute a db2mtrk command: db2mtrk -a -i -d. And when this command is executed, you might see something similar to the output shown below.

Listing 16. Output excerpt produced by the db2mtrk utility

Tracking Memory on: 2012/08/27 at 19:57:13 Memory for instance other fcmbp monh 9.5M 768.0K 320.0K Memory for database: SAMPLE utilh pckcacheh other catcacheh bph (1) bph (S32K) 64.0K 192.0K 128.0K 192.0K 8.2M 832.0K bph (S16K) bph (S8K) bph (S4K) shsorth lockh dbh 576.0K 448.0K 384.0K 0 16.7M 14.2M apph (15) apph (14) apph (13) apph (12) apph (11) apph (10) 64.0K 64.0K 64.0K 64.0K 64.0K 64.0K 64.0K apph (9) apph (8) apph (7) appshrh 64.0K 64.0K 64.0K 64.0K 256.0K Application Memory for database: SAMPLE appshrh 256.0K Memory for application 13 apph other 64.0K 192.0K Memory for application 12 apph other 64.0K 192.0K Memory for application 11 apph other 128.0K 192.0K Memory for application 10 apph other 64.0K 192.0K

The db2pd utility

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

Listing 17. Basic syntax for the db2pd command

db2pd &lt;-inst&gt; &amp;lt;-database [DatabaseName] ,... | -alldatabases&gt; &amp;lt;-everything&gt; &amp;lt;-full&gt;

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, the `db2pd` utility will collect information for all elements of all databases on all database partition servers that are local to the server. If `-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 filters as part of the `db2pd` command executed, including the following options:

- `-applications`
- `-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, to obtain information about the transaction log files associated with the SAMPLE database, you could execute a `db2pd` command like this: `db2pd -database SAMPLE -logs`. Then you might see something similar to the output shown below.

### Listing 18. Output excerpt produced by the db2pd utility

```
Database Partition 4294967295 -- Database SAMPLE -- Active -- Up 0 days 00:22:31 -- Date 08/27/2012 20:18:18
Logs: Current Log Number 0 Pages Written 8 Cur Commit Disk Log Reads 0 Cur Commit Total Log Reads 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 Log Chain ID 0 Current LSN 0x0000000004E28C7A Address StartLSN State Size Pages Filename 0xA2F97888 00000000004E20010 0x00000000 1000 1000 S0000000.LOG 0xA2F97048 0000000005208010 0x00000000 1000 1000 S0000001.LOG 0xA2F76598 00000000055F0010 0x00000000 1000 1000 S0000002.LOG
```

For more information about the `db2pd` command, refer to the [DB2 Command Reference](ibm.com/developerWorks/dm2/).
Conclusion

This tutorial was designed to introduce you to the set of monitoring tools available with DB2 10.1 and to show you how each is used to monitor how well (or how poorly) your database system is operating. Database monitoring is a vital activity that, when performed regularly, provides continuous feedback on the health of a database system. There are two distinct methods for monitoring operations in a database. The first involves monitoring table functions and administrative views, and allows you to view information that shows the state of various aspects of the database at a specific point in time. The second method utilizes event monitors, that can capture and log historical information as specific types of database events occur.

When invoked, monitoring table functions collect information from relevant DB2 monitoring elements that continuously accumulate data at the request, activity, and data object levels. Monitoring table functions employ the new lightweight monitoring infrastructure introduced in DB2 9.7, which was designed to avoid performance degradation effects on the database operation. Monitoring data and metrics can also be obtained by using an alternative set of SQL administrative views.

Event monitors provide a way to collect monitor data when events or activities occur that cannot be captured using monitoring table functions. Additionally, while monitoring elements begin collecting information once the database is activated in background processes, event monitors must be specifically created before they can be used.

Workload management allows you to attain a stable and predictable environment through automatic identification and management of work carried out on the data server. Monitoring of workload management is critical for validating the health and efficiency of the system. Monitoring table functions and event monitors are used to monitor workload management efficiency. Workload management monitoring table functions generally provide computed value data that is more statistical in nature. Workload management event monitors include the Activity, Threshold Violation, and Statistics event monitors.

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 indices 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 chosen for a particular SQL statement. However, Visual Explain can only be used to view explain snapshot data; to view explain data collected and written to the explain tables, the db2exfmt tool must be used.

The db2ls and db2val utilities allow you to list installed DB2 products and features, and validate DB2 installed copies. System tools also include the db2mtrk utility, designed to provide a complete report of the memory status for instances, databases, and applications.
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 or troubleshoot a database system (or any component of a database system). The db2pd utility is invoked by executing the `db2pd` command.
Related topics

- Read the Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611 to learn in-depth information about each of the concepts presented in this tutorial. This guide is compilation of topics from the DB2 10.1 documentation.
- Use the DB2 documentation in IBM Knowledge Center to find more details about each of the concepts presented in this tutorial.
- Use an RSS feed to request notification for the upcoming tutorials in this series. (Find out more about RSS feeds of developerWorks content.)
- You can learn more about monitoring DB2 from the DB2 10.1 Information Center.
- Attend or download the material for Information Management bootcamps.
- Now you can use DB2 for free. 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.

© Copyright IBM Corporation 2012
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)