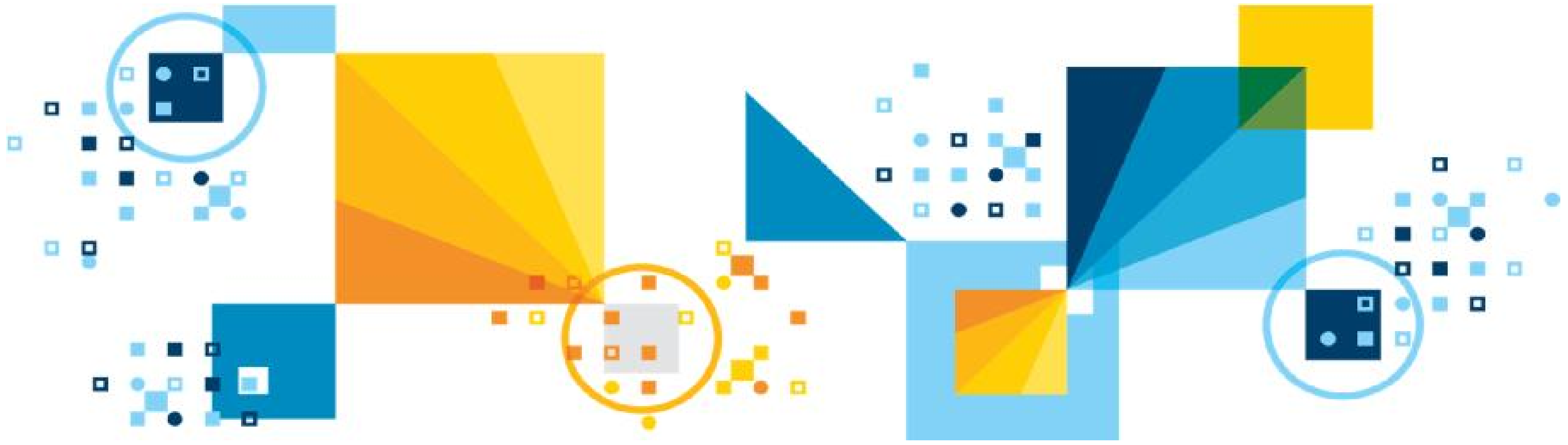
DB2 Performance Monitoring Essentials

Module ID	10301
Length	1.5 hours + 1.5 hour Hands on Lab



For questions about this presentation contact askdata@ca.ibm.com

February 9, 2015

PRESENTATION NOTES FOR PAGE 1

Disclaimer

PRESENTATION NOTES FOR PAGE 2

Module Information

- You should have completed or acquired the necessary knowledge for the following modules in order to complete this module:
  - Module Introduction to DB2 LUW Performance

- After completing this module, you should be able to:
  - Learn what to monitor, for both operational and exception situations enhancement in DB 10.5 and Optim Performance Manager

PRESENTATION NOTES FOR PAGE 3

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- Monitoring Interfaces Available with DB2
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 4

Learn what to monitor, for both operational and exception situations enhancement in DB 10 and Optim Performance Manager

Agenda

- **Introduction to Monitoring Essentials**
  - Learning objectives
  - Before we get started
- Monitor Elements
- Monitoring Interfaces Available with DB2
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 5

Learning Objectives

- Knowledge of new DB2 features to improve performance using monitoring targets
- Understand what monitoring options are available with DB2 and how to get started using them effectively
- DB2 monitoring framework and basic use of monitoring interfaces
- To identify system resource bottlenecks
- db2top tool
- Optim Performance Manager Tool

PRESENTATION NOTES FOR PAGE 6

Performance tuning for DB2 is a vast subject, with many components affecting performance and many approaches to monitoring and finding solutions. In this module we will focus on DB2 Monitoring Essentials.

First: We look at how and what to monitor on your DB2 system on a regular basis. A new monitoring infrastructure that is now part of DB2 is explored along with DB2 snapshot monitoring. This module will cover examples from both monitoring frameworks.

You will also learn about where to look for problems when they arise out of the blue.

Finally we examine system bottlenecks and how they can provide clues as to what DB2 activity might be contributing to system slow-down and what courses of action might be taken to alleviate such bottlenecks.

Introduction

§ **Database monitoring**
– Tasks associated with examining the operational status of your database

§ **Database monitoring is a vital activity for:**
– The maintenance of performance
– Health of your database management system

§ **Collects information from:**
– Database manager
– Its databases
– Connected applications

PRESENTATION NOTES FOR PAGE 7

Before We Get Started - DB2 LUW

§ **Runs on Operating Systems**
- AIX
- Linux
- zLinux
- Windows
- Solaris
- HP-UX

§ **Runs on Hardware vendors**

DB2 is DB2 all functionality at same level for all platforms

PRESENTATION NOTES FOR PAGE 8

Whether a new version of DB2 or a fixpak is released, it functions the same on all the major platforms that DB2 LUW runs on. DB2 runs on a rich variety of Operating systems making application solutions more portable in terms of infrastructure requirements. It comes in several major editions including a free Express-c version for developers, workgroup edition typically used in inter-departmental applications, and Enterprise Edition which represents the top of the line for features and functionality.

DB2 Runs On Multiple Platforms

- Challenge - OS performance tools/platform
- Good news: Many OS performance tools very similar across platforms
- More good news: OS architectures the same at abstract level
  - Most tuning methods/tools yield like results on all platforms
- More good news: DB2 monitoring elements and interfaces are the same on all platforms
- Best news: DB2 autonomics like STMM reduce the need for production tuning on all platforms

PRESENTATION NOTES FOR PAGE 9

There is a wealth of open source tools that can effectively be used to monitor system activity on a DB2 server and help identify system bottlenecks. Since DB2 runs on different platforms, coupled with the fact many system monitoring tools are open source, some of the tools available for system monitoring, like vmstat, iostat, top, etc, will be slightly different from system to system. System bottlenecks can help point you in the right direction when diagnosing tricky performance problems that can occur. The good news is that for the most part, most operating systems have similar architectures and rely on similar resources, their tools may be slightly different but the key metrics apply to all.

With very few exceptions, DB2 monitoring elements and interfaces are the same no matter which platform you are running DB2 on.

DB2 has embraced and delivered on the philosophy that autonomics, and self-tuning is an efficient and effective way to keep your DB systems responsive and optimized under changing conditions.

Good Foundation - Know Your System

- **What OS level is running ?**
- **How much swap space is set ?**
- **What are my processors speed ?**
- **How many processor cores are assigned ?**
- **What storage is present/available on the system?**
- **How much memory is available on the system after startup?**

Discovery Tools & Commands

- db2pd – osinfo (All)
- systeminfo (Windows)
- free (Linux)
- bootinfo (AIX)
- yast – (Linux)
- smit – (AIX)
- lvm – (AIX, Linux)

PRESENTATION NOTES FOR PAGE 10

One of the first things to do, as you first build any database server, is find out and record what exactly your system is and what resources are available for DB2 to utilize.

DB2 can automatically detect most resources and configure them accordingly, but it is always a good idea to identify what the system is, what it is running , how much memory it has, what kind of storage resources are available There are a number of tools that can provide the basic system information you'll want when configuring DB2 and it's components.

Agenda

- Introduction to Monitoring Essentials
- **Monitor Elements**
  - Types of data
  - Categories
  - Collection Levels
- Monitoring Interfaces Available with DB2
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 11

Database monitoring

The term database monitoring refers to the tasks associated with examining the operational status of your database.

Database monitoring is a vital activity for the maintenance of the performance and health of your database management system. To facilitate monitoring, DB2® collects information from the database manager, its databases, and any connected applications. With this information you can perform the following types of tasks, and more:

Forecast hardware requirements based on database usage patterns.

Analyze the performance of individual applications or SQL queries.

Track the usage of indexes and tables.

Pinpoint the cause of poor system performance.

Assess the impact of optimization activities (for example, altering database manager configuration parameters, adding indexes, or modifying SQL queries).

Monitor Elements

NEW IN DB2 10

§ **Data structures used to store information about a particular aspect of the database system status**

– Each monitor element reflects one of the following types of data:

- Counter: the number of times something happens
  - *deadlocks*: the total number of deadlocks that have occurred
  - *rows_deleted*: the number of row deletions attempted
  - *total_sorts*: the total number of sorts that have been executed
- Gauge: a measurement of how much of something is happening or is used
  - *total_section_proc_time* or *total_sort_time*: measures of how much time is used in different phases of processing
- Watermark: the highest value reached for a given measurement
  - *uow_total_time_top*: the lifetime of the longest-running unit of work since the database was activated
- Text: many monitor elements report text values
  - *stmt_text*: the text of an SQL statement
- Timestamp: the time that something happened
  - *conn_time*: the time that a connection was made to a database

PRESENTATION NOTES FOR PAGE 12

Monitor elements

Monitor elements are data structures used to store information about a particular aspect of the database system status. For example, the monitor element direct_reads reflects the number of read operations that take place that are performed directly from disk, rather than from any buffer pool.

Each monitor element reflects one of the following types of data: Counter Counters track the number of times something happens. For example, the deadlocks monitor element records the total number of deadlocks that have occurred. Other examples of counters include commit_sql_stmts (commit statements attempted), rows_deleted and total_sorts. Gauge Gauges reflect a measurement of how much of something is happening or is used. For example, time-spent monitor elements, such as total_section_proc_time or total_sort_time are measures of how much time is used in different phases of processing. Other examples of gauges include: locks_held, num_extent_moved, and sort_heap_allocated. Compared to counters, which can only increase over time, the values in gauges might go up or down, depending on what is happening in the database. Watermark Watermarks reflect the highest value reached for a given measurement. For example, uow_total_time_top shows the lifetime of the longest-running unit of work since the database was activated. Other examples of watermarks include: pkg_cache_size_top, and sort_heap_top. Text Many monitor elements report text values. For example, stmt_text contains the text of an SQL statement. Other examples of text monitor elements include: table_name, tablespace_type, and db_storage_path_state. Timestamp Timestamp monitor elements show the time that something happened. For example, conn_time shows the time that a connection was made to a database. Other examples of timestamp monitor elements include: lock_wait_start_time stmt_first_use_time, and uow_stop_time. Compared to gauges that measure elapsed time, timestamps measure the exact point in time that something begins or ends.

Monitor Elements - Categories

§ **Request monitor elements** (also known as *request metrics*)

- Measure the volume of work or effort expended by the database server to process requests issued directly by an external application (*application requests*), by a coordinator agent to a subagent or by an agent at a different database member
  - **Overall system processing**
    - *total_cpu_time*
    - *total_wait_time*
    - *total_rqst_time*
    - *rqsts_completed_total*
  - **Client-server processing**
    - *client_idle_wait_time*
    - *tcpip_recv_volume*
  - **Data Server processing**
    - *lock_wait_time*
    - *pool_read_time*
  - **Specific Data Server environment:**
    - *fcm_recv_wait_time*
    - *wlm_queue_time_total*
- Available through Table Functions and Event Monitors

PRESENTATION NOTES FOR PAGE 13

You can examine monitor elements using one or more of the various monitoring interfaces provided with the DB2® product, such as table functions or event monitors..

Request monitor elements
Request monitor elements, also known as request metrics, measure the volume of work or effort expended by the database server to process different types of requests, including overall system processing, requests related to a specific type of processing, and requests related to a specific data server environment.

Data object monitor elements
Data object monitor elements provide information about operations performed on particular data objects, including tables, indexes, buffer pools, table spaces, and containers.

Monitor element collection levels
The monitor element collection level for a monitor element refers to what, if any settings must be active for data to be collected for that element. For many monitor elements, data collection is controlled by configuration parameters, clauses in the DDL used to define workload management objects, or a combination of both.

Time-spent monitor elements
Time-spent monitor elements track how time is spent in the system. You can query them to see where time is spent waiting, or performing different types of processing. You can also view the elapsed time spent in a particular system component.

Logical data groups overview
When examining monitor element data, it is frequently useful to look at multiple related elements at the same time. Logical data groups are groupings of elements that complement each other.

Monitor element reference
A description of the data collected by the monitor element.

Monitor Elements - Categories

- **Activity monitor elements** (also known as *activity metrics*)
  - Subset of request monitor elements:
    - Monitor the work done to execute SQL statement sections, including locking, sorting, and row processing

      direct_read_time

      effective_isolation

      STMT_TEXT
  - Available through Table Functions and Event Monitors

- **Data Object monitor elements**
  - Provide information about operations performed on particular data objects, including tables, indexes, buffer pools, table spaces and containers

    TABLE_SCANS

    ROWS_INSERTED

    INDEX_SCANS
  - Available through Table Functions

NEW IN DB2 10

PRESENTATION NOTES FOR PAGE 14

You can examine monitor elements using one or more of the various monitoring interfaces provided with the DB2® product, such as table functions or event monitors..

Request monitor elements
Request monitor elements, also known as request metrics, measure the volume of work or effort expended by the database server to process different types of requests, including overall system processing, requests related to a specific type of processing, and requests related to a specific data server environment.

Data object monitor elements
Data object monitor elements provide information about operations performed on particular data objects, including tables, indexes, buffer pools, table spaces, and containers.

Monitor element collection levels
The monitor element collection level for a monitor element refers to what, if any settings must be active for data to be collected for that element. For many monitor elements, data collection is controlled by configuration parameters, clauses in the DDL used to define workload management objects, or a combination of both.

Time-spent monitor elements
Time-spent monitor elements track how time is spent in the system. You can query them to see where time is spent waiting, or performing different types of processing. You can also view the elapsed time spent in a particular system component.

Logical data groups overview
When examining monitor element data, it is frequently useful to look at multiple related elements at the same time. Logical data groups are groupings of elements that complement each other.

Monitor element reference
A description of the data collected by the monitor element.

Monitor Elements – Collection Levels

- **REQUEST METRICS {NONE | BASE | EXTENDED}**
  - Broadest (highest) collection level specified by:
    - Database configuration parameter: MON_REQ_METRICS
    - CREATE/ALTER SERVICE CLASS… COLLECT REQUEST METRICS
- **ACTIVITY METRICS {NONE | BASE | EXTENDED}**
  - Broadest (highest) collection level specified by:
    - Database configuration parameter: MON_ACT_METRICS
    - CREATE/ALTER WORKLAD… COLLECT ACTIVITY METRICS
- **DATA OBJECT METRICS {NONE | BASE | EXTENDED}**
  - Database configuration parameter: MON_OBJ_METRICS
- **Always collected**

PRESENTATION NOTES FOR PAGE 15

You can examine monitor elements using one or more of the various monitoring interfaces provided with the DB2® product, such as table functions or event monitors..

Request monitor elements
Request monitor elements, also known as request metrics, measure the volume of work or effort expended by the database server to process different types of requests, including overall system processing, requests related to a specific type of processing, and requests related to a specific data server environment.

Data object monitor elements
Data object monitor elements provide information about operations performed on particular data objects, including tables, indexes, buffer pools, table spaces, and containers.

Monitor element collection levels
The monitor element collection level for a monitor element refers to what, if any settings must be active for data to be collected for that element. For many monitor elements, data collection is controlled by configuration parameters, clauses in the DDL used to define workload management objects, or a combination of both.

Time-spent monitor elements
Time-spent monitor elements track how time is spent in the system. You can query them to see where time is spent waiting, or performing different types of processing. You can also view the elapsed time spent in a particular system component.

Logical data groups overview
When examining monitor element data, it is frequently useful to look at multiple related elements at the same time. Logical data groups are groupings of elements that complement each other.

Monitor element reference
A description of the data collected by the monitor element.

Monitor Elements – DB2 10 Enhancements

NEW IN DB2 10

§ **Some of the new monitor elements:**

- evmon_wait_time The amount of time that an agent waited for an event monitor record to become available
- total_extended_latch_wait_time The amount of time, in milliseconds, spent in extended latch waits
- total_extended_latch_waits The number of extended latch waits
- intra_parallel_state The current state of intrapartition parallelism reported at statement, activity, transaction, or workload level
- total_stats_fabrication_time Is the statistics collection activity needed to generate statistics during query compilation
- total_stats_fabrication_proc_time The total non-wait time spent on statistics fabrications by real-time statistics gathering
- total_sync_runstats_time The total time spent on synchronous RUNSTATS activities triggered by real-time statistics gathering
- total_disp_run_queue_time The total time that requests, that were run in this service class, spent waiting to access the CPU

PRESENTATION NOTES FOR PAGE 16

Monitoring Enhancements for Version 10.5

New Monitoring elements for new column-organized tables, for example

- Counters for total logical and physical column-organized data page reads and pages found
  - E,g, POOL_COL_L_READS, POOL_COL_P_READS
- Counter for column-organized data page writes:
  - POOL_COL_WRITES
- Counters for asynchronous column-organized data page reads and writes and pages found:
  - POOL_ASYNC_COL_READS,POOL_ASYNC_COL_WRITES
- Counters for column-organized data page reads per table
  - Object_COL_L_READS
  - Object_COL_P_READS
  - Object_COL_GBP_L_READS
  - Object_COL_GBP_P_READS

PRESENTATION NOTES FOR PAGE 17

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- **Monitoring Interfaces Available with DB2**
  - Monitor routines
    - Monitor (MON) Table Functions
    - Monitor (MON) Views
    - Event Monitors and Event Monitor (EVMON) Routines
  - Snapshot Monitoring
  - DB2 pureScale Monitoring
  - db2pd
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

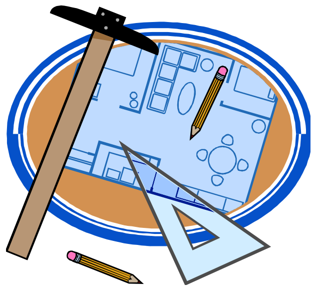
PRESENTATION NOTES FOR PAGE 18

Monitoring interfaces are the tools that let you peer into and measure how well DB2 is performing. We will look at the monitoring framework, snapshot monitoring, and event monitoring (which, depending on the event monitor you define can be working off the monitoring framework or the classical snapshot monitoring framework).

Lastly we will examine the db2pd, db2 problem determination tool, an extremely powerful and robust command interface that comes with each db2 installation.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine monitor elements and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2® use monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

Monitoring - Interfaces



- **Table Functions and Monitor Views**
  - Real-time monitoring accessible through SQL statements
  - Newer, lightweight, high-speed monitoring infrastructure
  - Evolved as complimentary extension to Workload Manager (WLM) implementation
  - Turn on collection at database level (more granularity available with WLM feature)
- **Event Monitors**
  - Capture information about database operations over time, as specific types of events that take place in your system
- **Snapshot Monitor**
  - Switch based monitoring
  - Services snapshot command, most Event Monitors, Administrative Views and Table Functions
  - Snapshots are useful for determining the status of a database system
- **DB2 problem determination tool a.k.a db2pd**

PRESENTATION NOTES FOR PAGE 19

In order to establish performance monitoring on your DB2 database system, you should first familiarize yourself with the various interfaces that are available.

The performance monitoring infrastructure and interfaces are at a crossroads.

A monitoring framework has been implemented that was developed as a complimentary extension of DB2’s Work Load Management Feature. It requires less overhead and offers more metrics than snapshot monitoring.

The snapshot monitor itself, is still part of the DB2 and still serves as a monitoring option. It is robust and well known.

The monitoring framework is the future basis of DB2 monitoring, and should be used going forward.

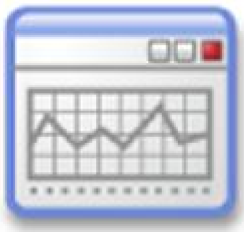
We will examine some of the db2’s monitoring interfaces available and their use, talk about what’s new, look at snapshot monitoring interfaces, and review some examples.

Then we well explore event monitors, where monitoring information is captured based on a specifications that define an event in an event monitor. A event monitor for locking has been introduced we will discuss its features and use.

To finish this section we will take a brief look at the DB2 Problem Determination Tool and how it can be used to help monitor performance on a DB2 system. It is a light weight powerful, broad scoped tool that can be used to investigate deep into the workings and resource allocations made on your DB2 installations.

Monitoring Framework - Three Focus Areas

- **System**
  - Provide total perspective of application work being done by database system
  - Aggregated through the WLM infrastructure
- **Activity**
  - Provide perspective of work being done by specific SQL statements
  - Aggregated through the package cache infrastructure
- **Data objects**
  - Provide perspective of impact of application work on data objects
  - Aggregated through data storage infrastructure



PRESENTATION NOTES FOR PAGE 20

Before we look at the new monitoring interfaces, more specifically the new monitoring table functions and views, lets consider the three different perspectives and collection points for the monitoring framework: System, Data Objects and Activity.

System – Here we find aggregations of total work being done, these are rolled up and defined along the components that make up the infrastructure of WLM, like workloads and service classes.

Data Objects – Consists of statistics of direct actions taken in the realm of data objects like tablespaces, bufferpools, containers…

Activity – Consists of statistics of activity associated with SQL statements.

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- **Monitoring Interfaces Available with DB2**
  - Monitor routines
    - **Monitor (MON) Table Functions**
    - Monitor (MON) Views
    - Event Monitors and Event Monitor (EVMON) Routines
  - Snapshot Monitoring
  - DB2 pureScale Monitoring
  - db2pd
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

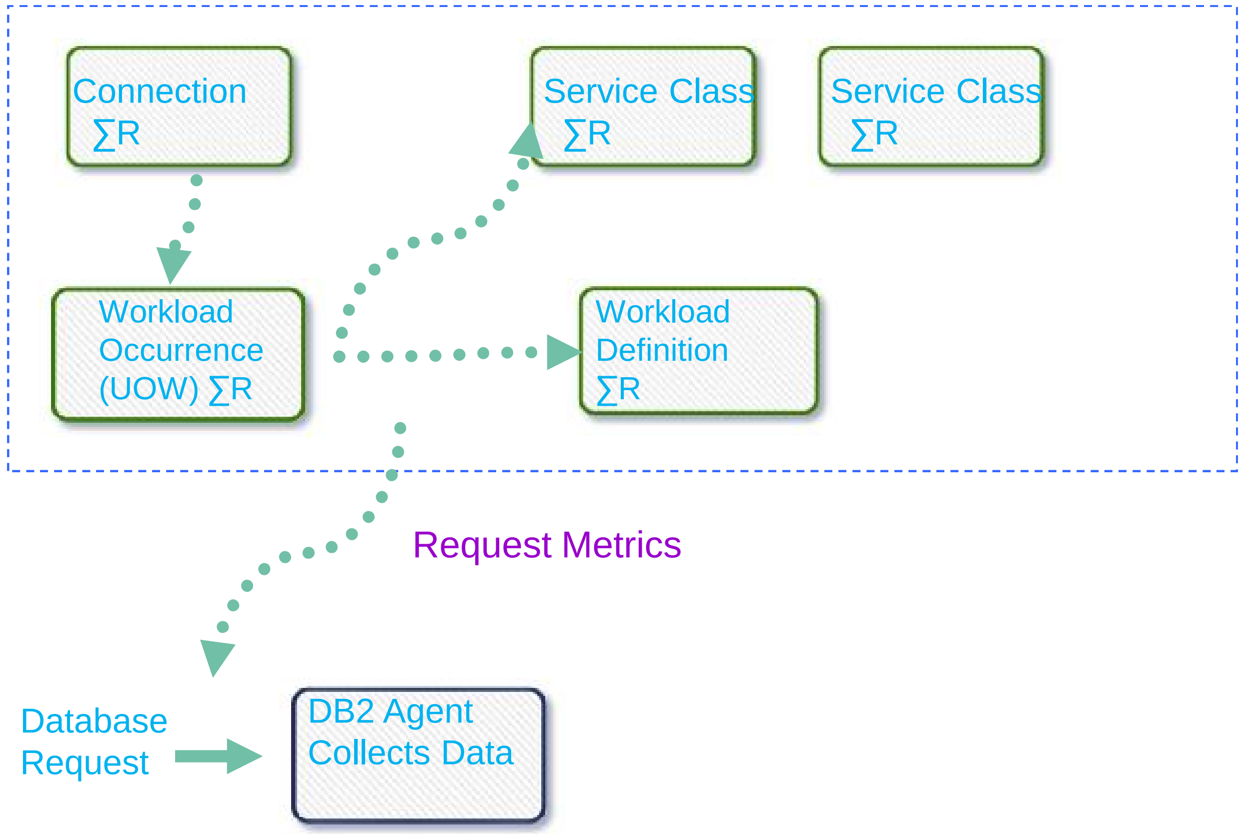
PRESENTATION NOTES FOR PAGE 21

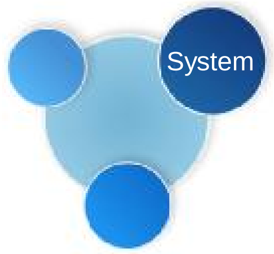
Monitoring interfaces are the tools that let you peer into and measure how well DB2 is performing. We will look at the monitoring framework, snapshot monitoring, and event monitoring (which, depending on the event monitor you define can be working off the monitoring framework or the classical snapshot monitoring framework).

Lastly we will examine the db2pd, db2 problem determination tool, an extremely powerful and robust command interface that comes with each db2 installation.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine monitor elements and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2® use monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

In-Memory Metrics :: System Perspective





Legend
ΣR = Accumulation of request metrics collected by agent

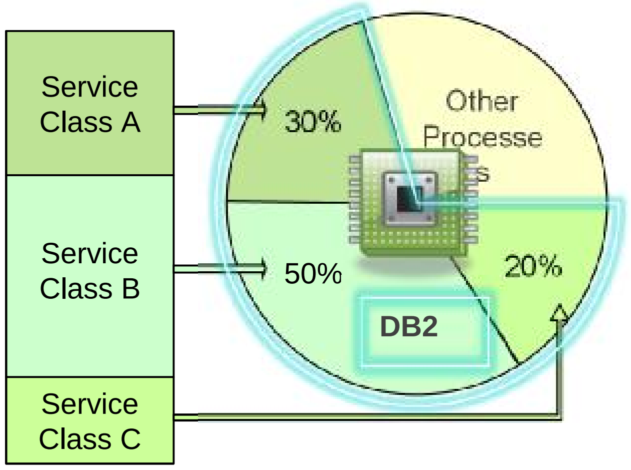
PRESENTATION NOTES FOR PAGE 22

A request comes into an agent and is processed. The agent gathers the metrics as it is running and at regular intervals propagates them upwards to the different aggregation points in the system hierarchy. By aggregating at regular intervals along both transactional and temporal boundaries, there is less of an imposition on the system when a “report” on such statistics is required. This is a different approach to snapshot monitoring where aggregation occurs at the time of a snapshot imposing considerable overhead upon the system.

Access Points :: System Perspective

System Monitoring Table Functions:

- **MON_GET_UNIT_OF_WORK**
  - Returns metrics for one or more units of work
- **MON_GET_WORKLOAD**
  - Returns metrics for one or more workloads
- **MON_GET_CONNECTION**
  - Returns metrics for one or more connections
- **MON_GET_SERVICE_SUBCLASS**
  - Returns metrics for one or more service subclasses
- **Also provide interfaces that produce XML output:**
  - MON_GET_UNIT_OF_WORK_DETAILS
  - MON_GET_WORKLOAD_DETAILS
  - MON_GET_CONNECTION_DETAILS
  - MON_GET_SERVICE_SUBCLASS_DETAILS



PRESENTATION NOTES FOR PAGE 23

Here are some of the interfaces, or access points that you can use to interrogate statistics that are aggregated at the system level.

Examples (usage):

The metrics returned by the MON_GET_UNIT_OF_WORK table function represent the accumulation of all metrics for requests that were submitted during a unit of work. Metrics are rolled up periodically during the unit of work. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an in interval of time, use the function to query the metric at the start and end of the interval, and compute the difference.

The metrics returned by the MON_GET_WORKLOAD table function represent the accumulation of all metrics for requests that were submitted by connections mapped to the identified workload object. Metrics are rolled up to a workload on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_WORKLOAD table function to query the metric at the start and end of the interval, and compute the difference.

The metrics returned by the MON_GET_CONNECTION table function represent the accumulation of all metrics for requests that were submitted by a connection. Metrics are rolled up at unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_CONNECTION table function to query the metric at the start and end of the interval, and compute the difference.

The metrics returned by the MON_GET_SERVICE_SUBCLASS table function represent the accumulation of all metrics for requests that have executed under the indicated service subclass. Metrics are rolled up to a service class on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_SERVICE_SUBCLASS table function to query the metric at the start and end of the interval, and compute the difference.

Access Points :: System Perspective

EXAMPLE:

§ **Display connections that return the highest volume of data to clients, ordered by rows returned**

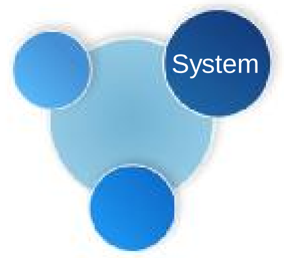
```
APPLICATION_HANDLE  ROWS_RETURNED     TCPIP_SEND_VOLUME   EVMON_WAIT_TIME    TOTAL_PEAS     TOTAL_CONNECT_REQUEST_TIME
------------------- -------------------- -------------------- -------------------- ------------------ ---------------------------
                 26                  436               341504                    0                  0                       101
                 20                   24                    0                    0                  0                         7
                 22                   16                 1174                    0                  0                        37
                 18                    1                 1422                    0                  0                       852
                  7                    1                    0                    0                  0                      4699
                 25                    0                67940                    0                  0                        10
                 24                    0               129842                    0                  0                         6
                 23                    0                97262                    0                  0                         8
```

```
db2 "SELECT application_handle
          , rows_returned
          , tcpip_send_volume
          , evmon_wait_time
          , total_peas
          , total_connect_request_time
       FROM TABLE(MON_GET_CONNECTION(NULL,-2)) AS t
      ORDER BY rows_returned DESC ";
```

PRESENTATION NOTES FOR PAGE 24

Access Points :: System Memory

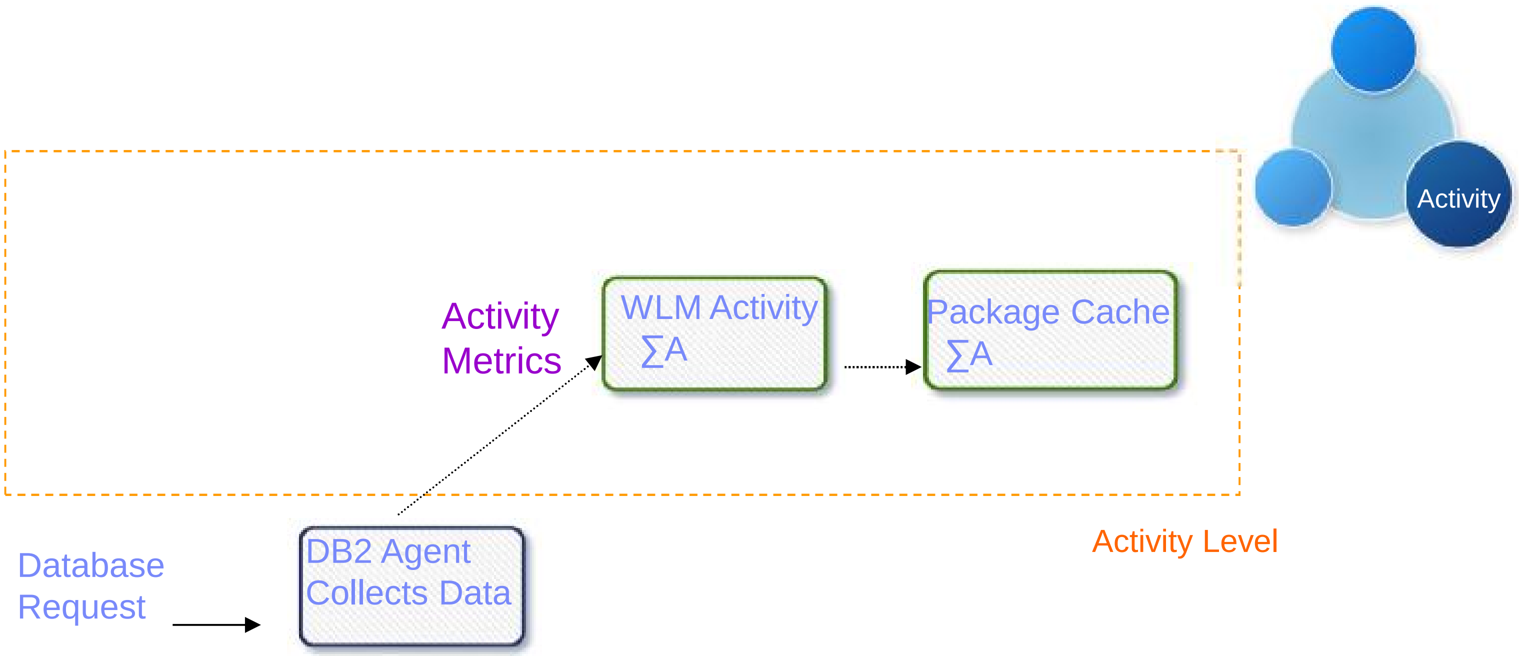
System Memory Monitoring Table Functions:



- **MON_GET_MEMORY_POOL**
  - Retrieves metrics from the memory pools contained within a memory set
- **MON_GET_MEMORY_SET**
  - Retrieves metrics from the allocated memory sets, both at the instance level and for all active databases within the instance
- **Other *Miscellaneous* Monitoring Table Functions:**
  - MON_GET_FCM
  - MON_GET_FCM_CONNECTION_LIST
  - MON_GET_EXTENT_MOVEMENT_STATUS

PRESENTATION NOTES FOR PAGE 25

In-Memory Metrics :: Activity Perspective



Legend
ΣA = Accumulation of metrics from activity execution portion of request

PRESENTATION NOTES FOR PAGE 26

A request comes into an agent and is processed. If the request is related to an activity, then the agent gathers the metrics from the start of activity execution and at regular intervals aggregates them in the activity control block. When the activity completes, those activity execution metrics are propagated to the package cache and aggregated under the specific cached section that was executed (static and dynamic SQL).

Access Points :: Activity Perspective

§ **Activities Monitoring Table Functions:**

- **MON_GET_PKG_CACHE_STMT**
  - Aggregates of the metrics gathered during each execution of a particular SQL statement (static or dynamic)
- **MON_GET_PKG_CACHE_STMT_DETAILS**
- **MON_GET_ACTIVITY_DETAILS**
  - Information on activities currently running on a system

PRESENTATION NOTES FOR PAGE 27

Here are some of the interfaces, or access points that you can use to interrogate statistics that are aggregated at the system level.

Examples (usage):

The metrics returned by the MON_GET_UNIT_OF_WORK table function represent the accumulation of all metrics for requests that were submitted during a unit of work. Metrics are rolled up periodically during the unit of work. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an in interval of time, use the function to query the metric at the start and end of the interval, and compute the difference.

The metrics returned by the MON_GET_WORKLOAD table function represent the accumulation of all metrics for requests that were submitted by connections mapped to the identified workload object. Metrics are rolled up to a workload on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_WORKLOAD table function to query the metric at the start and end of the interval, and compute the difference.

The metrics returned by the MON_GET_CONNECTION table function represent the accumulation of all metrics for requests that were submitted by a connection. Metrics are rolled up at unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_CONNECTION table function to query the metric at the start and end of the interval, and compute the difference.

The metrics returned by the MON_GET_SERVICE_SUBCLASS table function represent the accumulation of all metrics for requests that have executed under the indicated service subclass. Metrics are rolled up to a service class on unit of work boundaries, and periodically during the execution of requests. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup. Metrics are strictly increasing in value. To determine the value of a given metric for an interval of time, use the MON_GET_SERVICE_SUBCLASS table function to query the metric at the start and end of the interval, and compute the difference.

Access Points :: Activity Perspective

§ **MON_GET_PKG_CACHE_STMT**

– Returns a point-in-time view of both **static** and **dynamic** SQL statements in the database package cache

– EXAMPLE:
List all the dynamic SQL statements from the database package cache ordered by the average CPU time:

```
MEMBER SECTION_TYPE      AVG_CPU_TIME          LOCK_WAIT_TIME                     STMT_TEXT
------ ------------ -------------------- -------------------- ----------------------------------------
     0 D                              11                    0 SET CURRENT LOCK TIMEOUT 5
     0 D                             123                    0 INSERT INTO WE_F6QEE8J2Y.WE_0G0_18846 (v
     0 D                            1753                    0 SELECT STATS_FLAG FROM SYSTOOLS.HMON_ATM
     0 D                            1907                    0 INSERT INTO WE_F6QEE8J2Y.WE_0G0_17064 (v
     0 D                            1920                    0 DELETE FROM WE_F6QEE8J2Y.WE_0G0_18846 WH
     0 D                            2142                    0 SELECT TABSCHEMA, TABNAME FROM SYSCAT.TA
     0 D                            2248                    0 SELECT POLICY FROM SYSTOOLS.POLICY WHERE
     0 D                            2450                    0 UPDATE WE_F6QEE8J2Y.WE_0G0_18846 SET val
     0 D                           33623                    0 SELECT * FROM WE_F6QEE8J2Y.WE_0G0_3128 O
     0 D                           34477                    0 UPDATE WE_F6QEE8J2Y.WE_0G0_18536 SET val
     0 D                           34678                    0 DELETE FROM WE_F6QEE8J2Y.WE_0G0_4062 WHE
```

```
db2 "SELECT MEMBER
          , SECTION_TYPE
          , TOTAL_CPU_TIME/NUM_EXEC_WITH_METRICS as AVG_CPU_TIME
          , LOCK_WAIT_TIME
          , SUBSTR(STMT_TEXT,1,40) STMT_TEXT
     FROM TABLE(SYSPROC.MON_GET_PKG_CACHE_STMT('D',NULL,NULL,-2)) as T
    WHERE T.NUM_EXEC_WITH_METRICS <> 0
    ORDER BY AVG_CPU_TIME"
```

PRESENTATION NOTES FOR PAGE 28

MON_GET_PKG_CACHE_STMT

Table function returns a point-in-time view of both static and dynamic SQL statements in the database package cache.

An optional input argument (either "D" or "S") of type CHAR(1) that specifies information type for the returned statement. If the argument is NULL or the empty string, information is returned for all SQL statements. Not case sensitive: "D" stands for dynamic; "S" for static.

An optional input argument of type VARCHAR (32) for bit data that specifies a unique section of the database package cache. If a null value is specified, information is returned for all SQL statements. Note that when the executable_id is specified, the section_type argument is ignored. For example, if an executable_id is specified for a dynamic statement, the dynamic statement details will be returned by this table function even if section_type is specified as static ("S").

Note: It takes a longer time period to build the compilation environment and to transfer statement text (which can be as large as 2 MB) between members. To improve performance when retrieving a list of all the statements from the package cache, do not to select the STMT_TEXT and the COMP_ENV_DESC columns.

With the preceding output, the executable_id can be used to find out the details about the most expensive statement (in terms of the average CPU time):

Usage notes

The MON_GET_PKG_CACHE_STMT table function returns a point-in-time view of both static and dynamic SQL statements in the database package cache. This allows you to examine the aggregated metrics for a particular SQL statement, allowing you to quickly determine the reasons for poor query performance. The metrics returned are aggregates of the metrics gathered during each execution of the statement.

It also allows you to compare the behavior of an individual cached section, relative to the other statements, to assist in identifying the most expensive section or statements (in terms of the execution costs).

The activity metrics reported by this function are rolled up to the database cache at the end of the execution of the activity.

Metrics collection for the execution of any statement is controlled through the COLLECT ACTIVITY METRICS clause on workloads, or the mon_act_metrics database configuration parameter at the database level. Metrics are only collected for executions of the statement if the statement was submitted by a connection associated with a workload or database for which activity metrics are enabled. The num_exec_with_metrics element returned by the MON_GET_PKG_CACHE_STMT function indicates how many executions of the statement have had metrics collected and have contributed to the aggregate metrics reported. If no metrics are collected for any execution of the statement, then the num_exec_with_metrics element is 0 and all metric values are returned as 0.

Access Points :: Activity Perspective

§ MON_GET_ACTIVITY_DETAILS (XML)

- Returns details about an activity, including general activity information (like statement text) and a set of metrics for the activity
- EXAMPLE:

Captures information about all the activities currently running on a system:

```
APP...HANDLE  A..._ID  UOW_ID  T...ACT_TIME  T...WAIT_TIME  STMT_TEXT
------------  -------  ------  ------------  -------------  ----------------------------------------------
15            1        5       16            5              select name from sysibm.systables
15            1        3       17            5              select * from sysibm.systables
7             1        49      0             0              with A1 as (select * from ……….
3 record(s) selected with 1 warning messages printed.
```

```
WITH A1 AS
   (SELECT * FROM TABLE(wlm_get_workload_occurrence_activities(null, -1)) WHERE activity_id > 0)
SELECT A1.application_handle
     , A1.activity_id
     , A1.uow_id
     , A1.total_act_time
     , A1.total_act_wait_time
     , varchar(actmetrics.stmt_text, 50) AS stmt_text
  FROM A1
     , TABLE(MON_GET_ACTIVITY_DETAILS(A1.application_handle, A1.uow_id,A1.activity_id, -1)) AS ACTDETAILS
     , XMLTABLE ( XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon')
                , '$actmetrics/db2_activity_details'
                  PASSING XMLPARSE(DOCUMENT ACTDETAILS.DETAILS) AS "actmetrics“
                  COLUMNS "STMT_TEXT" VARCHAR(1024) PATH 'stmt_text‘
                        , "TOTAL_ACT_TIME" INTEGER PATH 'activity_metrics/total_act_time‘
                        , "TOTAL_ACT_WAIT_TIME" INTEGER PATH 'activity_metrics/total_act_wait_time'
               ) AS ACTMETRICS
```

PRESENTATION NOTES FOR PAGE 29

The MON_GET_ACTIVITY_DETAILS table function returns details about an activity, including general activity information (like statement text) and a set of metrics for the activity.

Table function parameters

application_handle An input argument of type BIGINT that specifies a valid application handle. If the argument is null, no rows are returned from this function, and an SQL0171N error is returned.

uow_id An input argument of type INTEGER that specifies a valid unit of work identifier unique within the application. If the argument is null, no rows are returned from this function, and an SQL0171N error is returned.

activity_id An input argument of type INTEGER that specifies a valid activity ID unique within the unit of work. If the argument is null, no rows are returned from this function, and an SQL0171N error is returned.

member An input argument of type INTEGER that specifies a valid member number in the same instance as the currently connected database when calling this function. Specify -1 for the current database member, or -2 for all database members. If the null value is specified, -1 is set implicitly.

Usage notes

The MON_GET_ACTIVITY_DETAILS function provides maximum flexibility for formatting output because it returns detailed information for a single activity as an XML document. The XML output includes both descriptive information (for example, statement text) and metrics. The output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function as shown in the example.

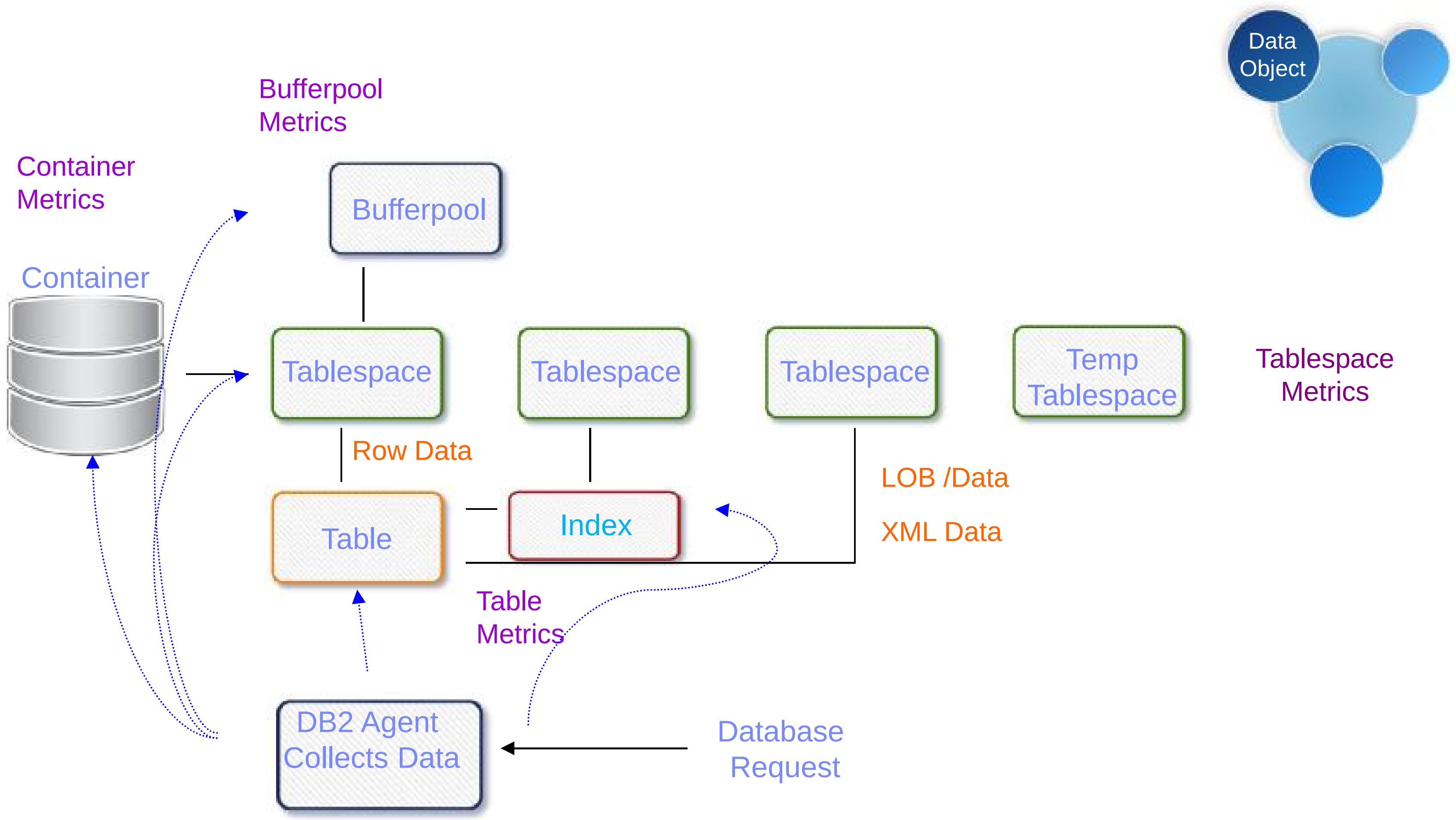
The metrics reported through this function (for example, CPU usage) are rolled up to the activity periodically during the lifetime of the activity. Therefore, the values reported by this table function reflect the current state of the system at the time of the most recent rollup.

Activity metrics are controlled through the COLLECT ACTIVITY METRICS clause on workloads, or the mon_act_metrics database configuration parameter at the database level. Metrics are collected if the connection that submits the activity is associated with a workload or database for which activity metrics are enabled. If activity metrics are not collected for an activity, all metrics are reported as 0.

The MON_GET_ACTIVITY_DETAILS table function returns one row of data for each member on which the activity exists. No aggregation across members is performed for the metrics. However, aggregation can be achieved through SQL queries.

The schema for the XML document that is returned in the DETAILS column is available in the file sqllib/misc/DB2MonRoutines.xsd. Further details can be found in the file sqllib/misc/DB2MonCommon.xsd.

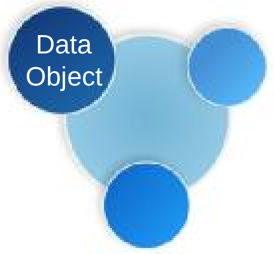
In-Memory Metrics :: Data Object Perspective



PRESENTATION NOTES FOR PAGE 30

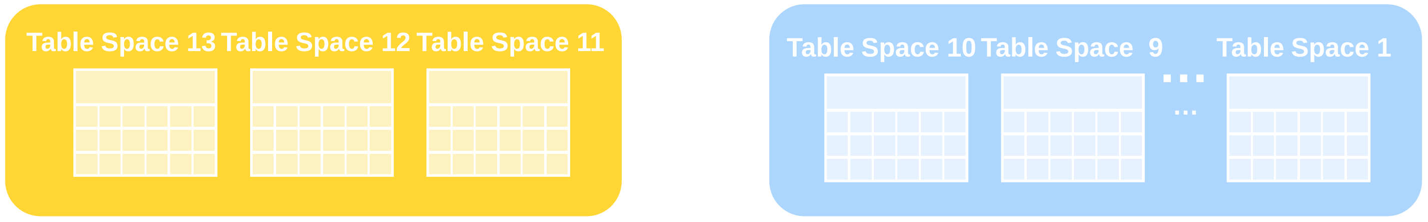
A request comes into an agent and is processed. As the request is processing, the agent increments the appropriate metrics at the different aggregation points in the object hierarchy.

Access Points :: Data Object Perspective



§ **Data Object Monitoring Table Functions:**

- **MON_GET_BUFFERPOOL**
  - Monitor bufferpool efficiency, hit ratio, activity
- **MON_GET_CONTAINER**
  - Monitor container activity, rank, enumerate
- **MON_GET_INDEX***
  - Monitor index usage, e.g. number of index scans, how many scans are index only scans
- **MON_GET_TABLE***
  - Monitor activity on table reads, updates, inserts, overflow activity
- **MON_GET_TABLESPACE**
  - Monitor tablespace activity (read and writes), bufferpool activity



***Always collected**

PRESENTATION NOTES FOR PAGE 31

As mentioned earlier there are three main collection streams that control the kind of metrics being made available, request metrics, activity metrics and object level metrics. The data object level metrics are well suited for monitoring the overall efficiency of your database and to find “hot spots”, in other words, where inputs and outputs are excessive to the point where they compromise performance. You can evaluate bufferpool, container, index, table, and tablespace activity, keeping tabs on and assessing your database’s performance.

The Data Object Monitoring Table functions and views are a good place to start working with the monitoring framework. They are similar in content to the existing administrative views and table functions, and accessible via SQL, and as such, easy to translate from snapshot monitoring functions and views.

Access Points :: Data Object Perspective

EXAMPLE:

- **List utilization of container file systems, ordered by highest utilization**

```
CONTAINER_NAME                                                    FS_ID       FS_USED_SIZE         FS_TOTAL_SIZE        UTILIZATION
----------------------------------------------------------------- ----------- -------------------- -------------------- ------------------
/home/db2inst1/db2inst1/NODE0000/DB2PT/T0000000/C0000000.CAT        2050          15056855040          19592417280         76.85
/home/db2inst1/db2inst1/NODE0000/DB2PT/T0000001/C0000000.TMP        2050          15056855040          19592417280         76.85
/home/db2inst1/db2inst1/NODE0000/DB2PT/T0000002/C0000000.LRG        2050          15056855040          19592417280         76.85
/home/db2inst1/db2inst1/NODE0000/DB2PT/T0000003/C0000000.LRG        2050          15056855040          19592417280         76.85
```

```
SELECT varchar(container_name, 65) as container_name
     , SUBSTR(fs_id,1,10) fs_id
     , fs_used_size
     , fs_total_size
     , CASE WHEN fs_total_size > 0
        THEN DEC(100*(FLOAT(fs_used_size)/FLOAT(fs_total_size)),5,2)
        ELSE DEC(-1,5,2)
       END as utilization
  FROM TABLE(MON_GET_CONTAINER('',-1)) AS t
 ORDER BY utilization DESC
```

PRESENTATION NOTES FOR PAGE 32

In-Memory Metrics :: Lock Perspective

§ **Lock Monitoring Table Functions:**

– **MON_GET_LOCKS**

- List of all locks in the currently connected database

– **MON_GET_APPL_LOCKWAIT**

- All locks that each application's agents, connected to the current database, are waiting to acquire

Always collected

PRESENTATION NOTES FOR PAGE 33

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- **Monitoring Interfaces Available with DB2**
  - Monitor routines
    - Monitor (MON) Table Functions
    - **Monitor (MON) Views**
    - Event Monitors and Event Monitor (EVMON) Routines
  - Snapshot Monitoring
  - DB2 pureScale Monitoring
  - db2pd
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 34

Monitoring interfaces are the tools that let you peer into and measure how well DB2 is performing. We will look at the monitoring framework, snapshot monitoring, and event monitoring (which, depending on the event monitor you define can be working off the monitoring framework or the classical snapshot monitoring framework).

Lastly we will examine the db2pd, db2 problem determination tool, an extremely powerful and robust command interface that comes with each db2 installation.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine monitor elements and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2® use monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

Monitor Views

- **MON_BP_UTILIZATION**
  - Buffer pool efficiency (e.g. hit ratios, average read and write times)
- **MON_CONNECTION_SUMMARY**
  - Incoming work per connection
  - New monitor elements: total_app_commits, total_app_rollbacks
- **MON_CURRENT_SQL**
  - Currently executing SQL statements (both static and dynamic)
- **MON_CURRENT_UOW**
  - Identify long running units of work and related activity
- **MON_DB_SUMMARY**
  - High level, aggregated metrics (percentage breakdowns, wait vs. active, totals)
- **MON_LOCKWAITS**
  - List applications currently waiting to acquire locks, holding applications, elapsed time, and statements
- **MON_PKG_CACHE_SUMMARY**
  - Aggregate metrics for statements currently in package cache, both dynamic and static
- **MON_SERVICE_SUBCLASS_SUMMARY**
  - Returns key metrics for all service subclasses in the currently connected database
- **MON_WORKLOAD_SUMMARY**
  - High level, aggregated metrics (percentage breakdowns, wait vs. active, totals)
- **MON_TBSP_UTILIZATION**
  - List tablespace information, state, high watermark, and hit ratios

PRESENTATION NOTES FOR PAGE 35

The monitoring framework is just that, new, and as such, improvements like additional monitoring elements and access points are being added quite frequently. These easy to use views have been added with fix pack 1.

##ADDITIONAL NOTES##

##

Notes on these views from IBM DB2 Version 10.1 Information Center 2012-04-30.

##

The MON_BP_UTILIZATION administrative view returns key monitoring metrics, including hit ratios and average read and write times, for all buffer pools and all database partitions in the currently connected database. It provides information that is critical for performance monitoring, because it helps you check how efficiently you are using your buffer pools.

The MON_CONNECTION_SUMMARY administrative view returns key metrics for all connections in the currently connected database. It is designed to help monitor the system in a high-level manner, showing incoming work per connection.

The MON_CURRENT_SQL administrative view returns key metrics for all activities that were submitted on all members of the database and have not yet been completed, including a point-in-time view of currently executing SQL statements (both static and dynamic) in the currently connected database.

The MON_CURRENT_UOW administrative view returns key metrics for all units of work that were submitted on all members of the database. It identifies long running units of work and can therefore be used to prevent performance problems.

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

The MON_LOCKWAITS administrative view returns information about agents working on behalf of applications that are waiting to obtain locks in the currently connected database. It is a useful query for identifying locking problems. This administrative view replaces the SNAPLOCKWAIT administrative view.

The MON_PKG_CACHE_SUMMARY administrative view returns key metrics for both static and dynamic SQL statements in the cache, providing a high-level summary of the database package cache. The metrics returned are aggregated over all executions of the statement across all members of the database.

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

The MON_TBSP_UTILIZATION administrative view returns key monitoring metrics, including hit ratios and utilization percentage, for all table spaces and all database partitions in the currently connected database. It provides critical information for monitoring performance as well as space utilization. This administrative view is a replacement for the TBSP_UTILIZATION administrative view.

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

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- **Monitoring Interfaces Available with DB2**
  - Monitor routines
    - Monitor (MON) Table Functions
    - Monitor (MON) Views
    - **Event Monitors and Event Monitor (EVMON) Routines**
  - Snapshot Monitoring
  - DB2 pureScale Monitoring
  - db2pd
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 36

Monitoring interfaces are the tools that let you peer into and measure how well DB2 is performing. We will look at the monitoring framework, snapshot monitoring, and event monitoring (which, depending on the event monitor you define can be working off the monitoring framework or the classical snapshot monitoring framework).

Lastly we will examine the db2pd, db2 problem determination tool, an extremely powerful and robust command interface that comes with each db2 installation.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine monitor elements and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2® use monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

Event Monitoring

- **To capture point-in-time information related to different kinds of events that take place in the system**
- **Created via SQL-DDL, definitions are stored in system catalog tables**
- **Output can be directed to:**
  - File
  - Table
  - Pipe

- **Types of events include:**
  - Locking
  - UOW
  - Statements (SQL)
  - Connections
  - Tables
- **New monitoring framework being used for new event monitors**
  - Some event monitors have been deprecated
    - DEADLOCKS + DETAILED DEADLOCKS à LOCKING
    - TRANSACTION à UNIT OF WORK (UOW)

PRESENTATION NOTES FOR PAGE 37

Event monitors are used to collect information about the database and any connected applications when specified events occur. Event monitors are database objects, and as such, they are created and manipulated using SQL data definition language (SQL DDL) statements.

To create an event monitor, use the CREATE EVENT MONITOR SQL statement. Event monitors collect event data only when they are active. To activate or deactivate an event monitor, use the SET EVENT MONITOR STATE SQL statement. The status of an event monitor (whether it is active or inactive) can be determined by the SQL function EVENT_MON_STATE.

Event monitor infrastructure provides an entirely set of monitor elements and methods to monitor DB2® events. As a result several existing event monitors have been deprecated and replacements introduced. Event monitors for Transactions have been deprecated, functionally replaced with a Event Monitor for Unit Of Work. The same is true for DEADLOCKS and DETAILEDDEADLOCK where a general LOCKING event monitor type has been introduced.

Types Of Events For Which Event Monitors Capture Data

NEW IN DB2 10

Type of event to monitor	Event monitor name	Details
Locks and deadlocks	LOCKING	To determine when locks or deadlocks occur, and the applications that are involved.
Execution of an SQL statement	ACTIVITIES	To capture activities for diagnostic reasons and to study the resource consumption of SQL
Execution of an SQL statement	STATEMENTS	To track what requests are being made to the database as a result of the execution of SQL statements
Completion of a unit of work (transaction)	UNIT OF WORK	To gather resource usage information and performance metrics for UOWs that run on the system
Eviction of sections from the package cache	PACKAGE CACHE	To capture a history of statements that are no longer in the package cache
Connections to the database by applications	CONNECTIONS	To capture metrics and other monitor elements for each connection to the database by an application
Deactivation of database	DATABASE	To capture metrics and other monitor elements that reflect information about the database as whole, since activation
Deactivation of database	BUFFERPOOLS	To capture metrics related to buffer pools
Deactivation of database	TABLESPACES	To capture metrics related to table spaces
Deactivation of database	TABLES	To capture metrics related to tables that have changed since database activation
Statistics and metrics on WLM objects	STATISTICS	To capture processing metrics related to WLM objects in the database
Exceeding a WLM threshold	THRESHOLD VIOLATIONS	To determine when specific thresholds that you set are exceeded during database operations
Changes to db or database manager configuration	CHANGE HISTORY	To capture change to db and db manager configuration and registry settings, execution of DDL statements and execution of utilities

PRESENTATION NOTES FOR PAGE 38

Types of events for which event monitors capture data

You can use event monitors to capture information related to many different kinds of events that take place on your system.

The following tables lists the types of events that occur in the system that you can monitor with an event monitor. It also describes the type of data collected for different events, as well as when the monitoring data is collected. The names of the event monitors shown in column two correspond to the keywords used to create that type of event monitor using the CREATE EVENT MONITOR statement.

Note

A detailed deadlock event monitor is created for each newly created database. This event monitor, named DB2DETAILDEADLOCK, starts when the database is activated and will write to files in the database directory. You can avoid the additional processor time this event monitor requires by dropping it. The DB2DETAILDEADLOCK event monitor is deprecated. Its use is no longer recommended and might be removed in a future release. Use the CREATE EVENT MONITOR FOR LOCKING statement to monitor lock-related events, such as lock timeouts, lock waits, and deadlocks

Working With Event Monitors - Procedure

1) Create the Event Monitor
- (Optional) Activate the Event Monitor

2) Enable the collection of data
- Only for:
  - LOCKING
  - ACTIVITIES
  - STATISTICS
  - UNIT OF WORK

3) Run your application or queries

4) (Optional) Deactivate the Event Monitor

5) Examine the data collected by the Event Monitor

6) (Optional) Prune data that is no longer needed from the Event Monitor Tables

PRESENTATION NOTES FOR PAGE 39

Working with event monitors

Generally, the process of creating and using event monitors to capture information about the system when certain events occur is similar for all event monitor types. First you create the event monitor, then you enable data collection, and finally, you access the data gathered.

Procedure

To use an event monitor to capture event information:

1.- Create the event monitor. To create an event monitor, use the appropriate version of the CREATE EVENT MONITOR statement. When you create an event monitor, you must choose how to record the data the event monitor collects. All event monitors can write their output to relational tables; however, depending on your specific purposes, there are different options that might be more appropriate.

2.- Activate the event monitor. To activate the event monitor, use the SET EVENT MONITOR STATE statement. For example, for an event monitor called capturestats, use the following command:SET EVENT MONITOR capturestats STATE 1 To turn off data collection by the event monitor, use the following statement:SET EVENT MONITOR capturestats STATE 0 By default, some event monitors activate automatically upon database activation; others require that you activate them manually. However, an event monitor created with the AUTOSTART option will not automatically be activated until the next database activation. Use the SET EVENT MONITOR STATE statement to force a recently-created event monitor into the active state. To determine whether an event monitor starts automatically, refer to the reference information for the relevant CREATE EVENT MONITOR statement

3.- Enable the collection of data. (Only for LOCKING, ACTIVITIES, STATISTICS, UNIT OF WORK and PACKAGE CACHE event monitors) Enabling data collection involves configuring the database manager to gather specific types of data to be recorded by event monitors.

Not all event monitors require data collection to be enabled; for those that do not, such as the TABLE event monitor, creating and activating them is sufficient to cause data to be collected. The threshold violations event monitor also starts data collection automatically; however, in this case, you must also define the thresholds for which you want data captured using the CREATE THRESHOLD statement.

For those event monitors that require data collection to be enabled, there are different options available to you. Depending on the type of event monitor you are working with, you might set a database configuration parameter to enable data collection across the entire database. Alternatively, you might choose to enable the collection of specific kinds of data for specific types of workload objects.

For example, to configure the collection of basic information for a unit of work event monitor when any unit of work in the system finishes, you can set the mon_uow_data parameter to BASE. Alternatively, to capture unit of work information only for a specific workload, you can specify the COLLECT UNIT OF WORK DATA BASE clause as part of the CREATE WORKLOAD or ALTER WORKLOAD statements.

4.- Run your applications or queries. After the event monitor has been created, and activated, and you have enabled data collection, run the applications or queries for which you want to collect data.

5.- Optional: Deactivate the event monitor. After you run the applications or queries for which you want data collected, you can deactivate the event monitor using the SET EVENT MONITOR STATE statement. (see step 2). Deactivating the event monitor is not necessary before proceeding to the next step, however leaving the event monitor active will result in disk space being used for data that you might not be interested in looking at.

6.- Examine the data collected by the event monitor. Depending on the type of output the event monitor creates, there are different options for accessing the data collected. If the data is written directly to a relational table, you can use SQL to access the data contained in the table columns. On the other hand, if the event monitor writes to an unformatted event (UE) table, you must post-process the UE table using a command like db2evmonfmt or a procedure like EVMON_FORMAT_UE_TO_TABLES before you can view the event data.

7.- Optional: Prune data that is no longer needed from the event monitor tables. For event monitors that you use on a regular basis, you might want to prune unneeded data from the tables. For example, if you use a unit of work event monitor to generate daily accounting reports about the system resources used by different applications, you might want to delete the current day's data from the event monitor tables once the reports have been generated.

Tip: If you need to prune event monitor output regularly, consider using an unformatted event (UE) table to record event monitor output. Starting in DB2® Version 10.1, UE tables can be pruned automatically after data is transferred to regular tables.

Working With Event Monitors – (1) Create the Event Monitor

NEW IN DB2 10

A. Determine type of Event Monitor

B. Decide type of output from the Event Monitor

Regular Tables

- ü Starting in DB2 Version 10, all event monitors can write output to regular tables
- ü Examine monitoring data at a later point in time
- ü Immediate access to data using SQL
- X CPU, log file, disk storage

Unformatted Event (UE) Tables

- ü Data written in binary format
- ü New PRUNE_UE_TABLE option for the procedure EVMON_FORMAT_UE_TO_TABLES, to prune data from the UE table after the extraction
- ü Examine monitoring data at a later point in time
- ü Performance, CPU, log file, disk storage
- X Require a post-processing operation to extract the data and to perform query using SQL

Files

- ü Managed by the Operating System
- ü Data stored outside of the database being monitored
- ü Examine the data offline at later point in time
- X SQL access

Named Pipes

- ü Output sent to a named pipe so the data can be used by another application immediately
- ü Event Data manipulation in real time
- X Access event data at a later point in time

PRESENTATION NOTES FOR PAGE 40

The ticks are the advantages for the Event monitor output type. The crosses are the disadvantages of the output type.
For example, “X SQL access” means no SQL access. “X CPU, log file, disk storage” means this type needs more CPU,
Log file, and disk storage.

Working With Event Monitors – (1) Create the Event Monitor

Type of event to monitor	Regular Table	UE Table	File	Named Pipe
LOCKING	ü	ü		
ACTIVITIES	ü		ü	ü
STATEMENTS	ü		ü	ü
UNIT OF WORK	ü	ü		
PACKAGE CACHE	ü	ü		
CONNECTIONS	ü		ü	ü
DATABASE	ü		ü	ü
BUFFERPOOLS	ü		ü	ü
TABLESPACES	ü		ü	ü
TABLES	ü		ü	ü
STATISTICS	ü		ü	ü
THRESHOLD VIOLATIONS	ü		ü	ü
CHANGE HISTORY	ü			

PRESENTATION NOTES FOR PAGE 41

Working With Event Monitors – (1) Create the Event Monitor

C. Issue a CREATE EVENT MONITOR statement

```
CREATE EVENT MONITOR evmon-name FOR eventtype
WRITE TO {TABLE | PIPE | FILE | UNFORMATTED EVENT TABLE}
{AUTOSTART | MANUALSTART}
```

- **`CONNECTIONS`** and **`STATEMENTS`** Event Monitors support the use of a `WHERE` clause on *application id, authorization id* and *application name*, in the `CREATE` or `ALTER EVENT MONITOR` statement
- **`BUFFERPOOLS`**, **`CONNECTIONS`**, **`DATABASE`**, **`STATEMENTS`**, **`TABLES`** and **`TABLESPACES`** Event Monitors can capture different types of events with a single event monitor definition

D. (Optional) If required by the type of event monitor created, activate it by issuing the SET EVENT MONITOR STATE statement

```
SET EVENT MONITOR evmon-name STATE 1
```

PRESENTATION NOTES FOR PAGE 42

Working With Event Monitors – (1) Create the Event Monitor

- **Recommended Practice:**
  - Table Space dedicated and configured to store the output table or tables associated with any event monitor
  - For Unformatted Event Table: create Table Spaces with at least an 8K pagesize to ensure that event data is contained within the inlined BLOB column of the UE table. If the BLOB column is not inlined, then the performance of writing and reading the events to the unformatted event table might not be efficient

PRESENTATION NOTES FOR PAGE 43

Working With Event Monitors – (2) Enable Data Collection

PASSIVE EVENT MONITORS:

§ **UNIT OF WORK**

– Database configuration parameter:
  - **MON_UOW_DATA** {NONE | BASE}
    MON_UOW_PKGLIST {OFF | ON}
    MON_UOW_EXECLIST {OFF | ON}

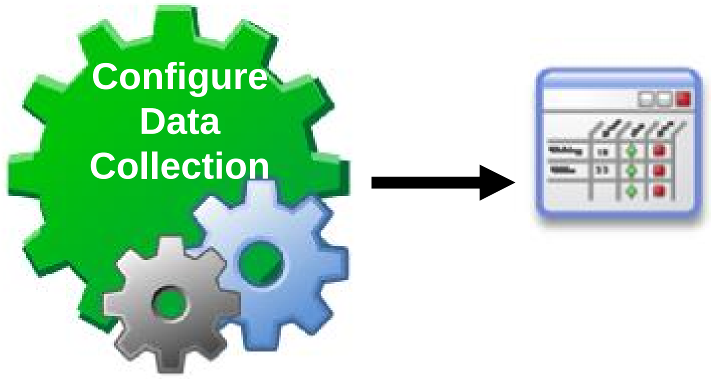
– CREATE/ALTER WORKLOAD … **COLLECT UNIT OF WORK DATA…**

– **N.B.**: Enable REQUEST METRICS collection

§ **LOCKING**

– Database configuration parameter:
  - **MON_LOCKWAIT**
  - **MON_LW_THRESH**
  - **MON_LOCKTIMEOUT**
  - **MON_DEADLOCK**

– CREATE/ALTER WORKLOAD …
  - **COLLECT LOCK WAIT DATA**
  - **COLLECT LOCK TIMEOUT DATA**
  - **COLLECT DEADLOCK DATA**



PRESENTATION NOTES FOR PAGE 44

Enabling event monitor data collection

Depending on the type of event monitor you are using, you might need to configure collection after you create the event monitor. By default, some event monitors collect certain data immediately when activated. Other event monitors require that you explicitly configure data collection independently of creating the event monitor. These types of event monitors are sometimes referred to as passive event monitors.

Before you begin

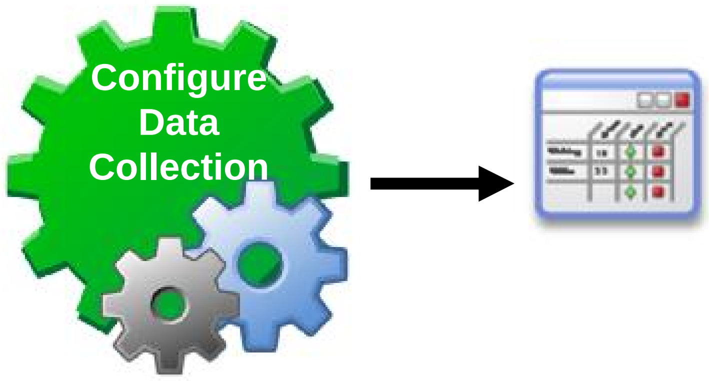
All event monitors must be activated before any data is written its target output table or tables (regular or UE), file or pipe. Some event monitors are configured by default as AUTOSTART event monitors. This means they are activated automatically when the database is activated. Others are configured by default to required that you activate them manually. Either way, you can override the default startup options. However, to start an automatic event monitor after you create it, but before the next database activation, you must use the SET EVENT MONITOR STATE statement to activate it manually.

Working With Event Monitors – (2) Enable Data Collection

PASSIVE EVENT MONITORS:

- **ACTIVITIES**
  - CREATE THRESHOLD… COLLECT ACTIVITY DATA…
  - CREATE/ALTER SERVICE CLASS…COLLECT ACTIVITY DATA…
  - CREATE/ALTER WORKLOAD… COLLECT ACTIVITY DATA…

- **STATISTICS**
  - Database configuration parameter
    - Enable REQUEST METRICS collection
  - CREATE/ALTER SERVICE CLASS … COLLECT REQUESTS METRICS…



PRESENTATION NOTES FOR PAGE 45

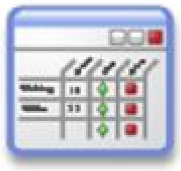
Working With Event Monitors – (4) Deactivate Event Monitor

- (Optional) Deactivate the Event Monitor by issuing the SET EVENT MONITOR STATE statement:

```
SET EVENT MONITOR evmon-name STATE 0
```

- Event Monitor Status:

```
SELECT EVMONNAME
     , EVENT_MON_STATE(EVMONNAME) STATUS
  FROM SYSCAT.EVENTMONITORS;
```



PRESENTATION NOTES FOR PAGE 46

About this task

Some event monitors support the use of a WHERE clause on the CREATE or ALTER EVENT MONITOR statement to capture event information selectively. The following event monitors, however, provide the ability to control what event data is collected independently of the event monitor definition: Activities

Change history

Locking

Statistics

Unit of work

Some of the event monitors listed collect certain types of data by default after the event monitor is activated; others require that you explicitly enable data collection. Either way, you can enable data collection in one of two ways, depending on the scope of activities for which you want data collected:

All activities in the database To have monitor data collected across all activities in the database, you modify the appropriate configuration parameter for the type of data you are interested in. For example, to have unit of work data collected for all units of work that run in the database, set mon_uow_data to BASE. In some cases, the default settings for configuration parameters are such that some type of data is always collected if there is an appropriate event monitor active to receive the date. For example, the default setting for mon_req_metrics is BASE; unless you override this setting, any active statistics or unit of work event monitor will record the values for the BASE set of request monitor elements.

Remember: Event monitors that support the use of the WHERE predicate collect only the data that satisfies the conditions specified in that predicate, regardless of the settings for any relevant configuration parameters.

Selected activities

Some event monitors - in particular, the workload management event monitors (threshold violations, statistics and activities) - provide the ability to control data collection for specific workload management objects. For example, you might choose to collect activity information for activities running in a specific service superclass. Configuring collection at this level generally involves adding a COLLECT clause to the CREATE or ALTER WORKLOAD (or SERVICE CLASS or WORK ACTION) statements to specify what type of information to collect for activities running under the auspices of that WLM object. For example, to enable the collection of extended statistics information for the service class urgent, you might use the following statement:

```
ALTER SERVICE CLASS urgent COLLECT AGGREGATE ACTIVITY DATA EXTENDED
```

Note: If a COLLECT clause is specified in a WLM CREATE or ALTER statement, the settings specified in the clause take precedence for that WLM object over any database-wide setting configured using a configuration parameter. For example, if mon_req_metrics is set to EXTENDED, and if workload payroll was configured to collect BASErequest metrics (for example, CREATE WORKLOAD payroll COLLECT REQUEST METRICS BASE), then extended request metrics are collected for all activities in the database except for the payroll workload.

Working With Event Monitors – (5) Examine the data collected by EM

§ **UNFORMATTED EVENT (UE) TABLE:**

- **db2evmonfmt** tool
  - Extracts data into a text report or into a formatted XML document
  - Limited capabilities to filter data (event ID, application, workload, …)
  - Setup and compilation of the Java source code provided (sqllib/samples/java/jdbc) is required before the tool can be used
  - Example:

```
java db2evmonfmt -f lock.xml -ftext -type lockwait -hours 5
```

- **EVMON_FORMAT_UE_TO_XML** table function
  - Extracts data into an XML document
  - PureXML features to query data
- **EVMON_FORMAT_UE_TO_TABLES** procedure
  - Extracts data into a set of relational tables
  - With PRUNE_UE_TABLES option, data that is successfully inserted into relational tables is deleted from the UE table

NEW IN DB2 10

PRESENTATION NOTES FOR PAGE 47

Enabling event monitor data collection

Depending on the type of event monitor you are using, you might need to configure collection after you create the event monitor. By default, some event monitors collect certain data immediately when activated. Other event monitors require that you explicitly configure data collection independently of creating the event monitor. These types of event monitors are sometimes referred to as passive event monitors.

Before you begin

All event monitors must be activated before any data is written its target output table or tables (regular or UE), file or pipe. Some event monitors are configured by default as AUTOSTART event monitors. This means they are activated automatically when the database is activated. Others are configured by default to required that you activate them manually. Either way, you can override the default startup options. However, to start an automatic event monitor after you create it, but before the next database activation, you must use the SET EVENT MONITOR STATE statement to activate it manually.

Working With Event Monitors – (5) Examine the data collected by EM

- **REGULAR TABLE:**
  - Run a **SELECT statement** to display the monitor element data
- **FILE or PIPELINE**
  - db2evmon command
    - Formats event monitor file and named pipe output, for display using standard output
    - EXAMPLE:

      Providing the path of the event files

      ```
      db2evmon -path '/tmp/dlevents'
      ```

      Providing the name of the database and the event monitor name

      ```
      db2evmon -db 'sample' -evm 'dlmon'
      ```

PRESENTATION NOTES FOR PAGE 48

Enabling event monitor data collection

Depending on the type of event monitor you are using, you might need to configure collection after you create the event monitor. By default, some event monitors collect certain data immediately when activated. Other event monitors require that you explicitly configure data collection independently of creating the event monitor. These types of event monitors are sometimes referred to as passive event monitors.

Before you begin

All event monitors must be activated before any data is written its target output table or tables (regular or UE), file or pipe. Some event monitors are configured by default as AUTOSTART event monitors. This means they are activated automatically when the database is activated. Others are configured by default to required that you activate them manually. Either way, you can override the default startup options. However, to start an automatic event monitor after you create it, but before the next database activation, you must use the SET EVENT MONITOR STATE statement to activate it manually.

Event Monitor Data Retention From Release To Release

NEW IN DB2 10

- **You can upgrade Event Monitor Output Tables after you upgrade the DB2 product**
  - To retain any data that might exist in Event Monitor Tables created before the upgrade
- **The `EVMON_UPGRADE_TABLES` procedure upgrades the definitions of existing Event Monitor (REGULAR and UE) Tables to the current level of DB2**
- **Use the `EVMON_FORMAT_UE_TO_TABLES` procedure with the `UPGRADE_TABLES` option to upgrade the set of Relational Tables produced from an UE table**
- **Implications of not upgrading event monitor tables**
  - Any new columns that have been added to the event monitor in the new release will not be populated with data, and will not available for queries
  - The values for any monitor elements that previously existed in the old release and that increased in size in the new release might be truncated

PRESENTATION NOTES FOR PAGE 49

Event monitor data retention from release to release

Beginning in DB2® Version 10.1, you can upgrade event monitor output tables after you upgrade the DB2 product. This capability lets you retain any data that might exist in event monitor tables that you had before you upgraded.

As event monitors are enhanced in the DB2 product, the tables they produce might change. For example, new columns might be added to a table for reporting new monitor elements. Before Version 10.1, if you had existing event monitors that wrote to tables that contained data that you wanted to retain, and you wanted to collect the data in the newly-added columns, you were required to manually alter them after upgrading to the new release. This alteration involved adding any of the new columns that you might want to use. If you did not add the new columns, the event monitor would work as it had in the previous release, capturing only the data supported by that the event monitor in that release.

Unformatted event tables that had changed could not be upgraded at all; you were required to drop them and then re-create them.

The EVMON_UPGRADE_TABLES stored procedure upgrades the definitions of existing event monitor tables to match those produced by the current level of the DB2 product. This feature lets you keep any existing tables that you might have, along with all the data they contain, eliminating the need to manually alter, or to drop and re-create tables.

Note: Starting in Version 10.1, you can also use the ALTER EVENT MONITOR statement to add new logical groups to an event monitor. You can use this approach as an alternative to EVMON_UPGRADE_TABLES to add logical data groups added in a new release. However, you cannot use ALTER EVENT MONITOR to modify logical groups that are already associated with the event monitor; if a logical data group already associated with the event monitor has changed, the only way to modify the event monitor is using the EVMON_UPGRADE_TABLES procedure.

The EVMON_UPGRADE_TABLES procedure works with both regular and UE tables. For regular tables, the procedure adds any new columns needed, drops old columns that are no longer required, and alters any columns as needed. For UE tables, the procedure adds new columns and modifies existing columns as needed to allow the UE table to be processed by the db2evmonfmt tool, or the EVMON_FORMAT_UE_TO_TABLES or EVMON_FORMAT_UE_TO_XML routines.

Important: Any active event monitors must be deactivated for the upgrade process to work properly. The EVMON_UPGRADE_TABLES procedure automatically deactivates any active event monitors before it begins upgrading tables. Do not reactivate any event monitors with tables being processed by EVMON_UPGRADE_TABLES, or the upgrade process will fail. Any event monitors that were active before the upgrade are activated again after the upgrade has completed.

Implications of not upgrading event monitor tables

As in past releases, you can choose to not upgrade your event monitor tables. However, any new columns that have been added to the event monitor in the new release will not be populated with data, and will not available for queries. Also, the values for any monitor elements that previously existed in the old release and that increased in size in the new release might be truncated. For example, if a monitor element increases in size from VARCHAR(20) to VARCHAR(128) in the new release, and you do not upgrade the previously-existing tables, the column that contains the monitor element values will continue to store only 20 characters of data, even though the system may be sending 128-bytes of data for that monitor element to the event monitor.

Upgrading tables produced by EVMON_FORMAT_UE_TO_TABLES

When used with UE tables, the EVMON_UPGRADE_TABLES procedure upgrades the UE table itself; it has no effect on any regular tables that you might have created using the EVMON_FORMAT_UE_TO_TABLES procedure. After you use EVMON_UPGRADE_TABLES to upgrade a UE table, you can also upgrade the output tables produced by EVMON_FORMAT_UE_TO_TABLES. As of DB2 Version 10.1 EVMON_FORMAT_UE_TO_TABLES procedure supports a new option: UPGRADE_TABLES. When you run the EVMON_FORMAT_UE_TO_TABLES procedure with this option, any existing tables produced by the procedure are altered so that the table columns match the output produced by the new version of EVMON_FORMAT_UE_TO_TABLES procedure.

Working With Event Monitors - Altering An Event Monitor

NEW IN DB2 10

- **LOGICAL DATA GROUPS**
  - Monitor Elements are grouped on logical data group
  - EXAMPLE:
    - ACTIVITIES logical data groups:

```
event_activity
event_activity_metrics
event_activitystmt
event_activityvals
```

- **An Event Monitor cannot be changed**
  - EXCEPTION: one or more logical data groups can be added* using
    `ALTER EVENT MONITOR … ADD LOGICAL GROUP…`
  - DEFAULT: all logical data groups that are associated with that Event Monitor are captured

```
CREATE EVENT MONITOR myacts FOR ACTIVITIES
WRITE TO TABLE ACTIVITY, ACTIVITYMETRICS;
ALTER EVENT MONITOR myacts
   ADD LOGICAL GROUP ACTIVITYSTMT
   ADD LOGICAL GROUP ACTIVITYVALS;
```

* Reactivation of the Event Monitor required to start to gather the new metrics

PRESENTATION NOTES FOR PAGE 50

Altering an event monitor

You cannot change an event monitor, with one exception: you can add one or more logical data groups to the set of logical data groups that the event monitor collects. You use the ALTER EVENT MONITOR statement to add logical groups.

About this task

When you create an event monitor that writes to tables, by default, all logical data groups of monitor elements that are associated with that event monitor are captured. However, if you include the names of logical data groups in the CREATE EVENT MONITOR statement, only those groups are captured. For example, you might create an activities event monitor that captures data only from the event_activity and event_activity_metrics logical data groups, as shown in the following example:

CREATE EVENT MONITOR myacts FOR ACTIVITIES WRITE TO TABLE event_activity, event_activity_metrics

The preceding DDL statement creates an event monitor that writes to two tables: ACTIVITY_myacts and ACTIVITY_METRICS_myacts.

Restrictions

You can use the ALTER EVENT MONITOR statement only to add logical data groups to an event monitor. You cannot remove a logical data group. You also cannot change the name, the target table space, or the value for PCTDEACTIVATE that is associated with the table that is used to capture the data in monitor elements that belong to a data group.

Procedure

To add additional logical data groups to an event monitor:

1.- Decide which logical data group you want to add. Using the preceding example of a locking event monitor where only two logical data groups are being captured, assume that you want to add the event_activitystmt and event_activityvals logical data groups.

2.- Formulate an ALTER EVENT MONITOR statement to add these new logical data groups.

ALTER EVENT MONITOR mylacts ADD LOGICAL GROUP event_activitystmt ADD LOGICAL GROUP event_activityvals

3.- Execute the statement.

Results

When the ALTER EVENT MONITOR statement completes execution, two additional tables are created for the event monitor myacts:ACTIVITYSTMT_myacts

ACTIVITYVALS_myacts

The next time the event monitor is activated, these tables are populated with data from their corresponding logical data groups.Remember: If you add new logical data groups to an event monitor, any data that existed for the logical data groups that were originally part of the table will not have any corresponding rows in the tables for the newly added logical group. Adjust your queries as needed, or consider pruning old data from the table after adding the logical groups.

Create New Event Monitor Example :: Statements

- EXAMPLE: Capture all statements where appl_id = myapp

```
CREATE EVENT MONITOR GET_SQL_MYAPP FOR STATEMENTS
WHERE (APPL_ID = 'myapp')
WRITE TO TABLE IN MYTBP PCTDEACTIVATE 70 BUFFERSIZE 8
AUTOSTART ;
```

- RECCOMENDATIONS:
  - For highly active event monitors use larger buffers
  - Place Event Monitor Tables in dedicated Tablespace
- It's possible use `db2evtbl` command to generate sample `CREATE EVENT MONITOR` SQL statement that write to SQL tables
- `SET EVENT MONITOR STATE 1` – to activate an Event Monitor
- `SET EVENT MONITOR STATE 0` – to deactivate an Event Monitor
- Use SQL to query results and find costly and error prone SQL statements
- CATALOG VIEWS
  - `EVENTS`
  - `EVENTMONITORS`
  - `EVENTTABLES`

PRESENTATION NOTES FOR PAGE 51

When you want to define an event monitor it can be done from a command line or you may use a graphical interface like DataStudio Administrator. Here we create an event monitor that will capture all statements where the connection attribute of appl_id = myapp and none other.   Eight 4k buffers have been defined to reduce I/O contention, it is also a good idea to create event monitoring tables in a dedicated tablespace that does not share disk resources with the database.

To create the tables that are written to by this event monitor, use the db2evtbl command, or the create event monitor gui in the control center is a very easy way to generate the ddl and tables for the event monitor output. Once the event monitor starts to collect data, these tables can be queried to flesh out High Cost / High Frequency SQL that may benefit from further tuning.

To get information about an event monitor the system catalog views for EVENTS, EVENTMONITORS, and EVENTTABLES can be queried.

Create New Event Monitor Example :: Locking

- **DB configuration parameters for collecting locking metrics (defaults)**

– Lock timeout events	(MON_LOCKTIMEOUT)	= NONE
– Deadlock events	(MON_DEADLOCK)	= WITHOUT_HIST
– Lock wait events	(MON_LOCKWAIT)	= NONE
– Lock wait event threshold	(MON_LW_THRESH)	= 5000000

- **TODO: Update default configurations:**

```
db2 update db cfg for SAMPLE using
  mon_lockwait    history
  mon_lw_thresh   3000000
  mon_locktimeout hist_and_values
  mon_deadlock    without_hist
```

– **WITHOUT_HIST**	– collect basic event information
– **HISTORY**	– collect up to 250 activities within same unit of work
– **HIST_AND_VALUES**	– collect activities and values
– **3000000**	– [µs], lock wait condition exists this long before lockwait

PRESENTATION NOTES FOR PAGE 52

In previous releases, if you wanted to monitor deadlock events, you had to issue the CREATE EVENT MONITOR FOR DEADLOCKS statement or check the output files for deadlock-related entries written by the automatically started DB2DETAILDEADLOCK event monitor. Includes a new event monitor infrastructure that provides an entirely new set of monitor elements and methods to monitor DB2® events. As a result, if you want to monitor deadlock events, using the CREATE EVENT MONITOR FOR LOCKING statement is the suggested method.

###ADDITIONAL NOTES##

The lock event monitor contains many additional monitor elements and is a more efficient method to collect lock-related events than the previous methods. The approach collects information about lock timeouts, deadlocks, and lock waits (that are more than a specified duration). The previous methods included using the DB2DETAILDEADLOCK event monitor, the DB2_CAPTURE_LOCKTIMEOUT registry variable, and the CREATE EVENT MONITOR FOR DEADLOCKS statement.

After using the lock event monitor to capture binary lock event data in an unformatted event table, you can convert the data into an XML or text document by using the Java™-based db2evmonfmt tool.

You can also choose to access the event monitor data through an XML document by using the new EVMON_FORMAT_UE_TO_XML table function, or through a relational table by using the EVMON_FORMAT_UE_TO_TABLES procedure.

Typically, you can achieve all your monitoring needs by creating a single event monitor per database for a given type of event (for example, lock events). You can alter settings to increase or decrease the amount of data that you can collect with the monitor to address changing monitoring needs. This contrasts with some older event monitors where a more common practice is to create a multiple event monitors, each geared to collect a particular monitoring need.

In this slide we take a look at the database configuration parameters used as controls to what and when the new event monitoring for locking is used and used for. Then you are shown the steps you would take to employ the new event monitor for locking, update what to collect via the mon_lockwait, mon_locktimeout, and mon_deadlock db configurtion parameters. The mon_lw_thresh (don’t ask me why lw ‘stead of lockwait, I suppose to be consistently inconsistent) qualifies as to how long of a wait constitutes and event, this value is in microseconds.

Create New Event Monitor Example :: Locking - Continued

- Designed to simplify the task of collecting locking data
- DB2DETAILDEADLOCK Event Monitor is DEPRECATED
  - Disable and remove it issuing the following SQL statements:

```
SET EVENT MONITOR DB2DETAILDEADLOCK state 0
DROP EVENT MONITOR DB2DETAILDEADLOCK
```

- Create Event Monitor that writes in an UE Table

```
CREATE EVENT MONITOR LOCKEVMON FOR LOCKING
WRITE TO UNFORMATTED EVENT TABLE
(TABLE IMRAN.LOCKEVENTS
     IN APPSPACE PCTDEACTIVATE 85)
```

PRESENTATION NOTES FOR PAGE 53

Lock and deadlock event monitoring

Diagnosing and correcting lock contention situations in large DB2® environments can be complex and time-consuming. The locking event monitor designed to simplify this task by collecting locking data.

The locking event monitor is used to capture descriptive information about lock events at the time that they occur. The information captured identifies the key applications involved in the lock contention that resulted in the lock event. Information is captured for both the lock requester (the application that received the deadlock or lock timeout error, or waited for a lock for more than the specified amount of time) and the current lock owner.

The information collected by the lock event monitor can be written in binary format to an unformatted event table in the database, in which case the captured data must be processed in a post-capture step. Alternatively, the lock event information can be written to a set of regular tables.

You can also directly access DB2 relational monitoring interfaces (table functions) to collect lock event information by using either dynamic or static SQL.

You must create a LOCK EVENT monitor using the CREATE EVENT MONITOR FOR LOCKING statement. You provide a name for the monitor and, if you are using UE tables as the output format, the name of an unformatted event table into which the lock event data is written.

Note: If you choose to use regular tables for event monitor output, default table names are assigned. You can override the defaults from the CREATE EVENT MONITOR statement, if you prefer.

LOCKING Specifies that this passive event monitor will record any lock event produced when DB2® runs into one or more of these conditions:

LOCKTIMEOUT: the lock has timed-out.

DEADLOCK: the lock was involved in a deadlock (victim and participant(s)).

LOCKWAIT: locks that are not acquired in the specified duration.

The creation of the lock event monitor does not indicate that the locking data will be collected immediately. The actual locking event of interest is controlled at the workload level or database level.

WRITE TO Specifies the target for the data.

TABLE Indicates that the target for the event monitor data is a set of formatted event tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

Notes

The target table is created when the CREATE EVENT MONITOR FOR LOCKING statement executes, if it doesn't already exist.

During CREATE EVENT MONITOR FOR LOCKING processing, if a table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR FOR LOCKING statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view. If the table exists and is not defined for use by another event monitor, then the event monitor will re-use the table.

Dropping the event monitor will not drop any tables. Any associated tables must be manually dropped after the event monitor is dropped.

Lock event data is not automatically pruned from either unformatted event tables or regular tables created by this event monitor. An option for pruning data from UE tables is available when using the EVMON_FORMAT_UE_TO_TABLES procedure. For event monitors that write to regular tables, event data must be pruned manually.

The FLUSH EVENT MONITOR statement is not applicable to this event monitor and will have no effect when issued against it.

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- **Monitoring Interfaces Available with DB2**
  - Monitor routines
    - Monitor (MON) Table Functions
    - Monitor (MON) Views
    - Event Monitor (EVMON) Routines
  - **Snapshot Monitoring**
  - DB2 pureScale Monitoring
  - db2pd
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 54

Monitoring interfaces are the tools that let you peer into and measure how well DB2 is performing. We will look at the monitoring framework, snapshot monitoring, and event monitoring (which, depending on the event monitor you define can be working off the monitoring framework or the classical snapshot monitoring framework).

Lastly we will examine the db2pd, db2 problem determination tool, an extremely powerful and robust command interface that comes with each db2 installation.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine monitor elements and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2® use monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

Snapshot Monitoring

- **Snapshot based monitoring - still viable**
- **Higher overhead than new monitoring framework**
- **Controlled with System Monitor switches**
  - DBM level vs SESSION level
  - Can reset counters at session level – easy to get current results
- **System Monitor data accessible through:**
  - Snapshot Monitor APIs (C or C++ application)
  - **GET SNAPSHOT** command (CLP)
    - For database manager, database, bufferpools, locks, dynamic SQL, applications, tables, tablespaces, etc. (the works)
    - Formatted text output by default – “great for greppers”
  - Snapshot Administrative Views and Snapshot Table Functions
    - Easy application interface to use
    - Easy to store in database relational tables
    - **SYSIBMADM.SNAP*** Views
    - **SYSPROC.SNAP_*** Table Functions

PRESENTATION NOTES FOR PAGE 55

You can use the snapshot monitor to capture information about the database and any connected applications at and over a specific time. Snapshots are useful for determining the status of a database system. Taken at regular intervals, they are also useful for observing trends and foreseeing potential problems. Some of the data from the snapshot monitor is obtained from the system monitor. The data available from the system monitor is determined by system monitor switches in the database management configuration file.

The system monitor accumulates information for a database only while it is active. If all applications disconnect from a database and the database deactivates, then the system monitor data for that database is no longer available. You can keep the database active until your final snapshot has been taken, either by starting the database with the ACTIVATE DATABASE command, or by maintaining a permanent connection to the database.

Snapshot data can be obtained with the get snapshot command for just about any and all database objects as well as with Administrative Views and Table Functions. The get snapshot commands output by default is in formatted text, if you are handy with the grep command it is fairly easy to extract the data that is important to your tuning efforts. Alternatively you can direct output to a file and query this output with the sqlcache_snapshot function.

Administrative Views and Table functions offer an easy way for an application to interface with snapshot performance metrics and an easy way to move snapshot data into tables for analysis.

Snapshot Monitoring - System Monitor Switch Control

- **Globally turned ON/OFF in DBM configuration (online)**

```
db2 get DBM CFG | grep DFT_MON
db2 update DBM CFG using
            DFT_MON_monitorswitch {ON | OFF}
db2 get snapshot for database on MYDB
```

- **Locally turned ON/OFF for the current SESSION**

```
db2 get MONITOR SWITCHES
db2 update MONITOR SWITCHES using
            monitorswitch {ON | OFF}
db2 get snapshot for database on MYDB
```

- **Reset counters**
  - At SESSION level: **db2 reset monitor all**
  - Globally: **DEACTIVATE/ACTIVATE DATABASE MYDB**

PRESENTATION NOTES FOR PAGE 56

Most performance metrics you will want to collect will require that the appropriate monitor switch is turned on. These switches can be turned on globally by updating the database manager configuration file and restarting the instance. These switches also can be controlled at the session level even if the global setting is turned off, but, of course, once the session is terminated so is the session's setting.

Another important thing to learn about, is how to reset the monitor switches to get a fresh set of metrics within a certain period of time when specific operations are being run. To do this for all monitor switches use the 'reset monitor all' command.

In the examples in the slide we see how to check the monitor switch settings in the dbm configuration file, how to update one of the switches, dft_mon_lock to off, we also see how it limits the amount of locking data available in a database snapshot. Then we turn on the switch at the session level and check to see that the Time database waited on locks is active.

Snapshot Monitoring - System Monitor Switches

- **Before capturing a snapshot or using an event monitor, you must determine what data you need the database manager to gather**
  - Buffer pool activity information
  - Lock, lock wait, and time related lock information
  - Sorting information
  - SQL statement information
  - Table activity information
  - Times and timestamp information
  - Unit of work information

Monitor Switch	DBM Parameter	Information Provided
BUFFERPOOL	DFT_MON_BUFPOOL	Number of reads and writes, time taken
LOCK	DFT_MON_LOCK	Lock wait times, deadlocks
SORT	DFT_MON_SORT	Number of heaps used, sort performance
STATEMENT	DFT_MON_STMT	Start/Stop time, statement identification
TABLE	DFT_MON_TABLE	Measure of activity(rows read/written)
UOW	DFT_MON_UOW	Start/end times, completion status
TIMESTAMP	DFT_MON_TIMESTAMP	Timestamps

PRESENTATION NOTES FOR PAGE 57

System monitor switches

System monitor switches control how snapshot monitors and some event monitors collect data.

The snapshot monitor and some event monitors report data collected by the system monitor. Collecting system monitor data introduces extra processing for the database manager. For example, in order to calculate the execution time of SQL statements, the database manager must make calls to the operating system to obtain timestamps before and after the execution of every statement. These types of system calls are generally expensive. The system monitor also increases memory consumption. For every monitor element tracked by the system monitor, the database manager uses its memory to store the collected data.

In order to minimize the additional processor time involved in maintaining monitoring information, monitor switches control the collection of potentially expensive data by the database manager. Each switch has only two settings: ON or OFF. If a monitor switch is OFF, the monitor elements under that switch's control do not collect any information. There is a considerable amount of basic monitoring data that is not under switch control, and will always be collected regardless of switch settings.

Each monitoring application has its own logical view of the monitor switches (and the system monitor data). Upon startup each application inherits its monitor switch settings from the dft_monswitches parameters in the database manager configuration file (at the instance level). A monitoring application can alter its monitor switch settings with the UPDATE MONITOR SWITCHES USING MONSWITCH OFF/ON command. The MONSWITCH parameter holds values found in the Monitor Switch column in the Snapshot Monitor Switches table shown in the next section. Changes to the switch settings at the application level only affect the application from where the switch was changed.

Instance-level monitor switches can be changed without stopping the database management system. To do this use the UPDATE DBM CFG USING DBMSWITCH OFF/ON command. The DBMSWITCH parameter holds values from the DBM Parameter column in the Snapshot Monitor Switches table shown in the next section. This dynamic updating of switches requires that the application performing the update be explicitly attached to the instance for the updates to dynamically take effect. Other existing snapshot applications will not be affected by a dynamic update. New monitoring applications will inherit the updated instance-level monitor switch settings. For an existing monitoring application to inherit the new default monitor switch values, it must terminate and re-establish its attachment. Updating the switches in the database manager configuration file will update the switches for all partitions in a partitioned database

The database manager keeps track of all the snapshot monitoring applications and their switch settings. If a switch is set to ON in one application's configuration, then the database manager always collects that monitor data. If the same switch is then set to OFF in the application's configuration, then the database manager will still collect data as long as there is at least one application with this switch turned ON.

The collection of time and timestamp elements is controlled by the TIMESTAMP switch. Turning this switch OFF (it is ON by default) instructs the database manager to skip any timestamp operating system calls when determining time or timestamp-related monitor elements. Turning this switch OFF becomes important as CPU utilization approaches 100%. When this occurs, the performance degradation caused by issuing timestamps increases dramatically. For monitor elements that can be controlled by the TIMESTAMP switch and another switch, if either of the switches is turned OFF, data is not collected. Therefore, if the TIMESTAMP switch is turned OFF, the overall cost of data under the control of other monitor switches is greatly reduced.

Event monitors are not affected by monitor switches in the same way as snapshot monitoring applications. When an event monitor is defined, it automatically turns ON the instance level monitor switches required by the specified event types. For example, a deadlock event monitor will automatically turn ON the LOCK monitor switch. The required monitor switches are turned ON when the event monitor is activated. When the event monitor is deactivated, the monitor switches are turned OFF.

The TIMESTAMP monitor switch is not set automatically by event monitors. It is the only monitor switch that controls the collection of any monitor elements belonging to event monitor logical data groupings. If the TIMESTAMP switch is OFF, most of the timestamp and time monitor elements collected by event monitors will not be collected. These elements are still written to the specified table, file, or pipe, but with a value of zero.

Snapshot Monitoring Examples - Database

§ **GET SNAPSHOT**

– Collects status information and formats the output for the user

– **get snapshot for**

- **`database on SAMPLE`**
- **`database manager`**
- **`application agentid #`**
- **`dynamic sql on SAMPLE`**

NEW IN DB2 10

§ **SNAPDB ADMINISTRATIVE VIEW**

– Allows you to retrieve snapshot information from the database (dbase) logical group for the currently connected database

§ **SNAP_GET_DB TABLE FUNCTION**

– Returns the same information as the SNAPDB administrative view

PRESENTATION NOTES FOR PAGE 58

GET SNAPSHOT command

Collects status information and formats the output for the user. The information returned represents a snapshot of the database manager operational status at the time the command was issued.

Scope

This command is invoked on the currently attached member and only returns information for that member by default. The currently attached member defaults to the host on which the command is run. If there are multiple members per host, the currently attached member is the first member listed in the db2nodes.cfg file on that host.

To run the command on a member that is not the current attach member, specify the AT MEMBER option to run this command on a specific member, or specify the GLOBAL option to run this command on all members to receive an aggregated result.

To change the current attach member from which to run any subsequent commands, use the ATTACH_MEMBER parameter of the SET CLIENT command. You must DETACH and then again ATTACH your application for this client setting to take effect.

Here are three different entry points to retrieve snapshot monitoring data. They are all getting the same information from the same source.

The get snapshot command works across a wide variety of data including database manager, database, bufferpools, tables and tablespaces. Its output is formatted text, by default, but can be redirected and parsed for querying, trending, and deriving ratio data.

The administrative views offer a SQL based application interface to this data. The administrative views are easy to use, they make calls to the administrative table functions and not suited for querying across partitions. They are more stable with regards to making coding changes against across different releases of DB2 compared to Administrative Table functions.

The administrative table functions, are more flexible than views, and better suited for gathering data across or on specific database partitions. They are more subject to revision between different releases of DB2.

Snapshot Monitoring Examples - Database

§ **Example: SNAPDB ADMINISTRATIVE VIEW**

– Retrieve the status, platform, location, and connect time for all database members of the currently connected database

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME
     , DB_STATUS
     , SERVER_PLATFORM
     , DB_LOCATION
     , DB_CONN_TIME
     , DBPARTITIONNUM
  FROM SYSIBMADM.SNAPDB
 ORDER BY DBPARTITIONNUM;
```

```
DB_NAME DB_STATUS SERVER_PLATFORM  DB_LOCATION        DB_CONN_TIME         DBPARTITIONNUM
------- --------- --------------- ------------ -------------------------- --------------
TEST       ACTIVE           AIX64        LOCAL 2006-01-08-16.48.30.665477              0
TEST       ACTIVE           AIX64        LOCAL 2006-01-08-16.48.34.005328              1
TEST       ACTIVE           AIX64        LOCAL 2006-01-08-16.48.34.007937              2
```

PRESENTATION NOTES FOR PAGE 59

This administrative view allows you to retrieve snapshot information from the dbase logical group for the currently connected database.

Used in conjunction with ADMIN_GET_STORAGE_PATHS table function, MON_GET_MEMORY_POOL, MON_GET_TRANSACTION_LOG, and MON_GET_HADR, the SNAPDB administrative view provides information equivalent to the GET SNAPSHOT FOR DATABASE on database-alias CLP command.

Snapshot Monitoring Examples - Database

- **Example: SNAP_GET_DB TABLE FUNCTION**
  - Retrieve the status, platform, location, and connect time as an aggregate view across all database members for all active databases in the same instance that contains the currently connected database

```
SELECT SUBSTR(DB_NAME, 1, 20) AS DB_NAME
     , DB_STATUS
     , SERVER_PLATFORM
     , DB_LOCATION
     , DB_CONN_TIME
  FROM TABLE(SNAP_GET_DB(CAST (NULL AS VARCHAR(128)), -2)) AS T
```

```
DB_NAME DB_STATUS SERVER_PLATFORM B_LOCATION          DB_CONN_TIME
------- --------- --------------- ------------ --------------------------
TOOLSDB    ACTIVE            AIX64 LOCAL        2005-07-24-22.26.54.396335
SAMPLE     ACTIVE            AIX64 LOCAL        2005-07-24-22.09.22.013196
```

PRESENTATION NOTES FOR PAGE 60

SNAP_GET_DB table function

The SNAP_GET_DB table function returns the same information as the SNAPDB administrative view.

Used in conjunction with the ADMIN_GET_STORAGE_PATHS table function, MON_GET_MEMORY_POOL, MON_GET_TRANSACTION_LOG, and MON_GET_HADR table functions, the SNAP_GET_DB table function provides information equivalent to the GET SNAPSHOT FOR ALL DATABASES CLP command.

Snapshot Monitor - db2top

- Most entries in snapshots are cumulative values and show the condition of the system at a point in time
- DB2TOP can be used to calculate the delta values for those snapshot entries in real time
- Run db2top in interactive mode

```
db2top –d SAMPLE
```

- Run db2top in batch mode

```
db2top -d SAMPLE -f collect.file -C -m 480 -i 15
db2top -d SAMPLE -f collect.file -b l -A
```

- **Object Monitored:**
  - Database (d)
  - Tablespaces (t)
  - Dynamic SQL (D)
  - Sessions (l)
  - Bufferpools (b)

PRESENTATION NOTES FOR PAGE 61

~~http://www.ibm.com/developerworks/data/library/techarticle/dm-0812wang/~~

```
Usage:  db2top [-d dbname] [-n nodename] [-u username] [-p password] [-V schema]
              [-i interval] [-P [part]] [-a] [-B] [-R] [-k] [-x]
              [-f file [+time] [/HH:MM:SS]]
              [-b options [-s [sample]] [-D separator] [-X] -o outfile]
              [-C] [-m duration]
       db2top -h

       -d : Database name (default DB2DBDFT)
       -n : Node name
       -u : User name
       -p : User password
       -V : Default explain schema
       -i : Interval in seconds between snapshots
       -b : background mode
            option: d=database, l=sessions, t=tablespaces, b=bufferpools, T=tables,
                    D=Dynamic SQL, s=Statements, U=Locks, u=Utilities, F=Federation,
                    m=Memory -X=XML Output, -L=Write queries to ALL.sql,
                    -A=Performance analysis
       -o : output file for background mode
       -a : Monitor only active objects
       -B : enable bold
       -R : Reset snapshot at startup
       -k : Display cumulated counters
       -x : Extended display
       -P : Partition snapshot (number or current)
       -f : Replay monitoring session from snapshot data collector file,
            can skip entries when +seconds is specified
       -D : Delimiter for -b option
       -C : Run db2top in snapshot data collector mode
       -m : Max duration in minutes for -b and -C
       -s : Max # of samples for -b
       -h : this help
```

Parameters can be set in $HOME/.db2toprc, type w in db2top to generate the resource configuration file.

`db2top -d sample -f collect.file -b l -A`

Option -A enables automatic performance analysis. So, the above command will analyze the most active sessions, which takes up the most CPU usage.

The following command runs db2top in replay mode, jumping to the time of interest to analyze.

`db2top -d sample -f collect.file /HH:MM:SS`

For example, the user restarts db2top in replay mode and it jumps to 2am exactly:

`db2top -d sample -f collect.file /02:00:00`

then, the user enters l to analyze what the session was doing.

Snapshot Monitoring vs. Monitoring Framework

- **Many monitoring elements are available through both types of monitoring**
  - Examples:
    - `pool_data_p_reads`: Indicates the number of data pages read in from the table space containers (physical) for regular and large table space
    - `lock_waits:` Returns lock waits information from an application snapshot
- **More monitoring elements in new framework & more to come**
- **More granularity in new framework** (actual vs. wait time)
- **Overall less overhead with new monitoring framework**
- **Avoid mixing frameworks when monitoring a system**
  - You will incur overhead from both

PRESENTATION NOTES FOR PAGE 62

Before breaking the interfaces down into separate topics, let’s start with a brief comparison of some of the attributes of Snapshot monitoring v. Monitoring framework and emphasize the ramifications of running both.

The monitoring framework runs with less overhead, more monitoring elements and more granularity in terms of what is monitored. Its interfaces have quickly caught up to those of snapshot monitoring.

Snapshot monitoring is the only monitoring framework available on older versions.

Pick one, we recommend getting to know about the monitoring framework and making use of it as soon as practical.

Avoid using both frameworks because you will incur overhead of both.

Monitoring Comparison

Feature	Monitoring Frame Work	Snapshot Monitoring	Event Monitoring
Extension of WLM	X		
Information captured based on a specification			X
Less overhead	X		X
More metrics	X		
Well known		X	X
Is the future basis of DB2 monitoring	X		
More granularity	X		
System Perspective	X		
Data Object Perspective	X		
Activity Perspective	X		
Information about the Database	X	X	X
Information about the Application	X	X	X
Controlled with switch		X	X
Store in relational tables		X	X
Table functions & views	X	X	
Active Database (required)		X	X

PRESENTATION NOTES FOR PAGE 63

Monitoring Enhancements

NEW IN DB2 10

§ **More comprehensive with higher granularity of control**

– New **CHANGE HISTORY** Event Monitoring tracks:
  - Changes on DB and DBM Configuration settings
  - Changes on Registry settings
  - Execution of DDL
  - Execution of Utilities

– New **`CREATE THRESHOLD`** statement domain
  - Now a threshold can be defined on statements that contain specific text

– New TABLE FUNCTIONS
  - **`MON_GET_CF_CMD`**
  - **`MON_GET_CF_WAIT_TIME`**
  - **`MON_GET_GROUP_BUFFERPOOL`**

– Two new SCALAR FUNCTIONS
  - **`MON_GET_APPLICATION_HANDLE`**
  - **`MON_GET_APPLICATION_ID`**

PRESENTATION NOTES FOR PAGE 64

New event monitor tracks configuration and registry changes and DDL and utility executions

The change history event monitor captures changes to database and database manager configuration and registry settings, execution of DDL statements, and execution of utilities. You can use this data to determine if the appearance of any problems with your database coincides with any of these events.

Some examples of changes to the system that might affect the performance or behavior of work on the system include:

Creating or dropping an index unexpectedly

The failure of scheduled maintenance to run

Changes to a database configuration parameter or DB2® registry setting

Execution of DDL statements

Execution of utilities (for example, RUNSTATS. LOAD, REORG)

STATEMENT domain for thresholds lets you define thresholds for statements that contain specific text

A new threshold domain called STATEMENT has been added to the CREATE THRESHOLD statement syntax. This domain makes it possible to define thresholds for execution of specific statements.

For example, you can define a CPUTIME threshold for a SQL statement like "SELECT * FROM TABLE1, TABLE2" so that a threshold violation occurs when this statement is executed and the CPU time threshold for the statement is exceeded. You can identify the statement for these thresholds by specifying either the statement text, as in this example, or the executable ID for the statement. Similar to thresholds on other domains, you can configure STATEMENT thresholds to write information about activities that violate the threshold to the activity event monitor

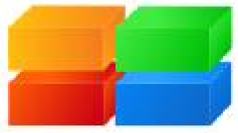
New and changed functions for accessing monitoring information

Several table functions and two scalar functions have been added and several table functions have been extended so that you can retrieve additional monitoring information using SQL.

Monitoring Enhancements - Continued

NEW IN DB2 10

- **USAGE LIST database object identifies statements that affect tables or indexes**
  - To record the DML statement sections that reference a particular table or index
  - To capture statistics about how DML statement sections affect each object as they execute
  - Includes information about:
    - The number of times that a section executed over a particular time frame
    - Aggregate statistics that indicate how the DML statement section affected the table or index across all executions
  - For each DML statement section, includes statistics about factors such as:
    - Lock
    - Buffer pool usage



Statistics

PRESENTATION NOTES FOR PAGE 65

Use the new usage list database object to record the DML statement sections that reference a particular table or index and capture statistics about how those sections affect each object as they execute.

Each entry in the usage list includes information about the number of times that a section executed over a particular time frame. The entries also contain aggregate statistics that indicate how the section affected the table or index across all executions.

The usage list also includes statistics about factors such as lock and buffer pool usage for each statement section. If you determine that a statement negatively affected a table or index, use these statistics to determine where further monitoring might be required or how you can tune the statement.

Monitoring Enhancements - Continued

NEW IN DB2 10

§ **STATEMENT domain for thresholds**

– Lets you define thresholds for statements that contain specific text
– Makes it possible to define thresholds for execution of specific statements
– You can define a CPUTIME threshold for a SQL statement
– You can quickly collect and examine activity information related to just that statement
– You can view the data represented by parameter markers that indicate a product identifier in the statement

PRESENTATION NOTES FOR PAGE 66

This new capability provides greater granularity and specificity in capturing information than in earlier releases. In previous releases, identifying problems with activities for a specific statement required that you capture information for many activities, and then sift through the event monitor data to look for anomalies

Monitoring Enhancements - Continued

NEW IN DB2 10

- **New Monitor Elements provide added insight into the operation of your DB2 server**
  - The operation of I/O servers (prefetchers)
  - The status of non-nested activities that are submitted by applications
  - Information about DATATAGINSC thresholds
  - Information about storage groups
  - Workload monitoring information
  - Time spent during connection and authentication activities
  - Details that are related to the longest-running SQL statement in the package cache
  - Additional measures of time spent in the system
  - Buffer pool and group buffer pool activity in DB2 pureScale environments
  - Information about usage lists
  - Information about memory pool and memory set usage

PRESENTATION NOTES FOR PAGE 67

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- **Monitoring Interfaces Available with DB2**
  - Monitor routines
    - Monitor (MON) Table Functions
    - Monitor (MON) Views
    - Event Monitor (EVMON) Routines
  - Snapshot Monitoring
  - **DB2 pureScale Monitoring** ⬅
  - db2pd tool
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 68

Monitoring interfaces are the tools that let you peer into and measure how well DB2 is performing. We will look at the monitoring framework, snapshot monitoring, and event monitoring (which, depending on the event monitor you define can be working off the monitoring framework or the classical snapshot monitoring framework).

Lastly we will examine the db2pd, db2 problem determination tool, an extremely powerful and robust command interface that comes with each db2 installation.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine monitor elements and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2® use monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

DB2 pureScale - Monitoring

- The DB2 **pureScale** feature provides a robust, highly available database processing environment
- **Monitoring**
  - **Gives a picture of how effectively the cluster is functioning**
  - **Interfaces:**
    - Table functions
    - Administrative views
    - Command line processor
  - **Returned information:**
    - **Member state**

      started, stopped, error, etc.
    - **CF state**

      primary, peer, catchup, error, etc.
    - **Alerts**

      indicate that there is something that requires further investigation
    - **AND MUCH MORE!!**

PRESENTATION NOTES FOR PAGE 69

The IBM® DB2® pureScale® Feature provides a robust, highly available database processing environment. Problems that might arise with the operation of one or more host systems in the DB2 pureScale instance can typically be addressed without interrupting access to data. Ironically, this characteristic of high-availability in a DB2 pureScale environment can mask issues that might lead to less than optimal performance. Monitoring certain aspects of your DB2 pureScale environment can help you recognize and address such issues.

DB2 pureScale - Monitoring Interfaces and Commands

- **Table functions and administrative views**
  - **DB2_GET_CLUSTER_HOST_STATE and DB2_CLUSTER_HOST_STATE**
    - Provides basic information about the hosts in DB2 pureScale instance
  - **DB2_GET_INSTANCE_INFO, DB2_MEMBER and DB2_CF**
    - Returns information about the role in the instance each host plays (cluster caching facility or member)
  - **DB2_INSTANCE_ALERTS**
    - provides information about alerts in the DB2 pureScale instance
- **Commands**
  - **LIST INSTANCE**
    - Returns information about the state of members, hosts, and cluster caching facilities
  - **LIST INSTANCE SHOW DETAIL**
    - Same as below plus partition number and connection information for members, hosts, and cluster caching facilities
  - **db2instance -list**
    - Reports the information about the hosts, members and CFs in the DB2 pureScale instance
  - **db2cluster -cm -list -alert**
    - Lists any alerts for cluster elements

PRESENTATION NOTES FOR PAGE 70

Monitoring – CF Memory & CPU Monitoring

- **Memory usage monitoring**
  - Group buffer pool memory
  - Lock memory
  - Shared Communication Area (SCA) memory
  - Overall CF memory
- **CPU load**
  - You can view the overall CPU load for the host serving as the CF in a DB2 pureScale instance using the ENV_CF_SYS_RESOURCES administrative view
- **Note: The value reported for CPU load reflects the total usage of the CPU for actual processing performed by the CF, and for host processes other than processes from the CF**

PRESENTATION NOTES FOR PAGE 71

Agenda

- Introduction to Monitoring Essentials
- Monitor Elements
- **Monitoring Interfaces Available with DB2**
  - Monitor routines
    - Monitor (MON) Table Functions
    - Monitor (MON) Views
    - Event Monitor (EVMON) Routines
  - Snapshot Monitoring
  - DB2 pureScale Monitoring
  - **db2pd tool**
- Essential Monitoring Targets
- Performance Monitoring Tools
- Summary

PRESENTATION NOTES FOR PAGE 72

Monitoring interfaces are the tools that let you peer into and measure how well DB2 is performing. We will look at the monitoring framework, snapshot monitoring, and event monitoring (which, depending on the event monitor you define can be working off the monitoring framework or the classical snapshot monitoring framework).

Lastly we will examine the db2pd, db2 problem determination tool, an extremely powerful and robust command interface that comes with each db2 installation.

You can monitor your database operations in real-time using monitoring table functions. For example, you can use a monitoring table function to examine the total amount of space used in a table space. These table functions let you examine monitor elements and metrics that report on virtually all aspects of database operations using SQL. The monitoring table functions use the newer, lightweight, high-speed monitoring infrastructure. In addition to the table functions, snapshot monitoring routines are also available. The snapshot monitoring facilities in DB2® use monitoring infrastructure. Generally speaking, snapshot monitoring facilities are no longer being enhanced in the product; where possible, use the monitoring table functions to retrieve the data you want to see.

db2pd - DB2 Problem Determination Tool

- **db2pd**
  - Diagnose lockwaits
  - Map application to dynamic SQL statement
  - Monitor Memory usage
  - Map application usage to tablespace
  - Monitor log usage
  - Includes WLM metrics
  - Assess Index usage
  - And more…

NEW IN DB2 10

- **The storage groups parameter has been added**
  - Provides details about storage groups and storage paths
  - If you do not specify a storage group identifier, information about all storage groups is reported

PRESENTATION NOTES FOR PAGE 73

Perhaps, one of the most under utilized tools for performance monitoring is the db2 problem determination tool. It is a bit verbose, in part due to its shear versatility. consider just some of the boons this tool has to offer, simply by mastering a bit of syntax.

Two benefits to collecting information without latching include faster retrieval and no competition for engine resources.

It is therefore possible (and expected) to retrieve information that is changing while db2pd is collecting information; hence the data might not be completely accurate. If changing memory pointers are encountered, a signal handler is used to prevent db2pd from ending abnormally. This can result in messages such as "Changing data structure forced command termination" to appear in the output. Nonetheless, the tool can be helpful for troubleshooting. Two benefits to collecting information without latching include faster retrieval and no competition for engine resources.

###

The following is a collection of examples in which the db2pd command can be used to expedite troubleshooting:

Example 1: Diagnosing a lockwait

Example 2: Using the -wlocks parameter to capture all the locks being waited on

Example 3: Using the -apinfo parameter to capture detailed runtime information about the lock owner and the lock waiter

Example 4: Using the callout scripts when considering a locking problem

Example 5: Mapping an application to a dynamic SQL statement

Example 6: Monitoring memory usage

Example 7: Determine which application is using up your table space

Example 8: Monitoring recovery

Example 9: Determining the amount of resources a transaction is using

Example 10: Monitoring log usage

Example 11: Viewing the sysplex list

Example 12: Generating stack traces

Example 13: Viewing memory statistics for a database partition

db2pd Example :: Mysterious Slowdown Every Afternoon

§ **A few of the many capabilities of db2pd**

§ **db2pd**

– Users report an overall slowdown every afternoon at 1pm, claiming nothing has changed

– Performing runstats does not solve the problem, eliminating out of date statistics as a culprit

– Checked different database indicators such as buffer pool hit ratios, lock waits etc. No bottlenecks found

– “**vmstat**” report shows high user cpu and first clue in the mystery:

```
procs -----------memory--------------- ----swap-- ------io----- -----system-- ------cpu------
 r  b     swpd    free    buff    cache  si     so     bi     bo     in     cs  us sy id wa st
 2  0   520800   60916    3800   438032  32    132    393    183    280    451  53  8 33  6  0
 1  0   520800   60636    3808   438032   0      0      0     48    259    376  57  4 37  2  0
 3  0   520800   60652    3824   438032   0      0      0      8    263    393  54  8 34  4  0
 1  0   520800   60776    3832   438032   0      0      0     26    258    371  60  1 39  0  0
```

PRESENTATION NOTES FOR PAGE 74

Let's run through an example of how db2pd can be used to diagnose performance issues. For starters, we have a slow down that is reported, regularly say in the afternoon, there are no abends or specific errors surfacing. Typically you might try some common fixes to common problems like updating statistics, checking the bufferpool hit ratio, look at locks, but in our scenario none of these efforts prove fruitful.

One clue we obtain is a higher than normal user CPU time being registered in VMSTAT.

Procs

- r: The number of processes waiting for run time.
- b: The number of processes in uninterruptible sleep.

Memory

- swpd: the amount of virtual memory used.
- free: the amount of idle memory.
- buff: the amount of memory used as buffers.
- cache: the amount of memory used as cache.
- inact: the amount of inactive memory. (-a option)
- active: the amount of active memory. (-a option)

Swap

- si: Amount of memory swapped in from disk (/s).
- so: Amount of memory swapped to disk (/s).

IO

- bi: Blocks received from a block device (blocks/s).
- bo: Blocks sent to a block device (blocks/s).

System

- in: The number of interrupts per second, including the clock.
- cs: The number of context switches per second.

CPU

- These are percentages of total CPU time.
- us: Time spent running non-kernel code. (user time, including nice time)
- sy: Time spent running kernel code. (system time)
- id: Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time.
- wa: Time spent waiting for IO. Prior to Linux 2.5.41, shown as zero.

db2pd Example :: Mysterious Slowdown Every Afternoon

- **User cpu is high which indicates that it is due to some activity in the application space (DB2) and not in the operating system**
- **Use "db2pd –edus" to see cpu usage by each thread:**

```
EDU ID      TID              Kernel TID       EDU Name          USR (s)       SYS (s)
…………………….
21          2946493344       10227            db2agent (DS2)    2.200000      0.900000
20          2947541920       10226            db2lfr (DS2)      0.000000      0.000000
19          2948590496       10225            db2loggw (DS2)    1.700000      0.600000
18          2949639072       10224            db2loggr (DS2)    2.170000      1.680000
17          2950687648       10214            db2agent (DS2)  29.310000      4.000000
16          2951736224       10201            db2resync         0.000000      0.000000
15          2952784800       10200            db2tcpcm          0.310000      0.450000
```

- **EDU id 17 stands out because it is using significantly more user cpu**
- **Please note: These values are cumulative over the life of EDU so it may be important to know the relative rate change of the CPU usage**

PRESENTATION NOTES FOR PAGE 75

The db2pd with the –edus option divulges, on a thread level which DB2 threads are using the lions share of the CPU. Here we see that EDU 17 is using quite a bit of CPU. EDU stands for Engine Dispatchable Unit.

-edus parameter

For the -edus parameter, the following information is returned:

EDU ID

The unique identifier for the engine dispatchable unit (EDU). Except on Linux operating systems, the EDU ID is mapped to the thread ID. On Linux operating system the EDU ID is a DB2 generated unique identifier.

TID

Thread identifier. Except on Linux operating systems, the thread ID is the unique identifier for the specific thread. On Linux operating systems, this is a DB2 generated unique identifier.

Kernel TID

A unique identifier for the operating system kernel thread in service.

EDU Name

DB2 specific name for the EDU.

USR

Total CPU user time consumed by the EDU.

SYS

Total CPU system time consumed by the EDU.

USR Delta

Indicates the delta of the CPU user time across a specified time interval.

SYS Delta

Indicates the delta of the CPU system time across a specified time interval.

db2pd Example :: Mysterious Slowdown Every Afternoon

- **Use "db2pd -db DS2 -applications" to find out the application being served by EDU ID 17:**

```
Address     App-      [nod-index]  Num-   Coor- Status          L-Anch- L-Stmt- Appid
            Handl                 Agents EDUID                    ID      UID
0x20A55FD0     12     [000-00012]   1      30   ConnectCompleted     0        0  *LOCAL.DB2.091015124129
0x20A50060     11     [000-00011]   1      29   ConnectCompleted     0        0  *LOCAL.DB2.091015124128
0x20A45FD0     10     [000-00010]   1      28   ConnectCompleted     0        0  *LOCAL.DB2.091015124127
0x20A40060      9     [000-00009]   1      27   ConnectCompleted     0        0  *LOCAL.DB2.091015124126
0x20A06C00      8     [000-00008]   1      26   ConnectCompleted     0        0  *LOCAL.DB2.091015124125
0x20A00060      7     [000-00007]   3      17   PerformingLoad     937        1  *LOCAL.johnh.091015130001
```

**** Some columns are not shown to fit the output**

- **Application handle 7 is being serviced by EDU id 17**
- **Status filed shows that it is performing a LOAD which explains the problem**
- **Novice user john has loads scheduled to run at 1pm**
- **Load is a resource intensive operation and running it during production hours causes everything else to run slower!**

PRESENTATION NOTES FOR PAGE 76

We here use db2pd to drill down into the applications that are running. We can now see that application handle 7, being serviced by EDU 17 as an agent number is performing a load, now hopefully most systems are subject to controls that would prevent a load from being introduced without its impact being tested, but it is possible, or perhaps a schedule was changed and not reviewed, what ever the case most systems are not fully protected against this type of thing happening.

-applications parameter

For the -applications parameter, the following information is returned:

ApplHandl

The application handle, including the node and the index.

NumAgents

The number of agents that are working on behalf of the application.

CoorPid

The process ID of the coordinator agent for the application.

Status

The status of the application.

Appid

The application ID. This value is the same as the appl_id monitor element data. For detailed information about how to interpret this value, see the documentation for the appl_id monitor element.

ClientIPAddress

The IP address from which the database connection was established.

EncryptionLvl

The data stream encryption used by the connection. This is one of NONE, LOW or HIGH. NONE implies that no data stream encryption is being used. LOW implies that the database server authentication type is set to DATA_ENCRYPT. HIGH implies that SSL is being used.

SystemAuthID

This is the system authorization ID of the connection.

ConnTrustType

The connection trust type. This is one of: non-trusted, implicit trusted connection, or explicit trusted connection.

TrustedContext

The name of the trusted context associated with the connection if the connection is either an implicit trusted connection or an explicit trusted connection.

RoleInherited

This is the role inherited through a trusted connection, if any.

Summary

§ In this Module you learned about:
- – DB2 monitoring infrastructure
- – What DB2 data to monitor
- – DB2 monitoring interfaces
- – Essential monitoring targets

PRESENTATION NOTES FOR PAGE 77

The next steps…

PRESENTATION NOTES FOR PAGE 78

The Next Steps…

- Complete the Hands on Lab for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module
  - Download the workbook 10301_WB1_DB2_Setup_BLUdb.pdf and the virtual machine image 10300_VM1_ DB2_PerformanceMonitoringAndTuning_10.5.part#.rar
  - Follow the instructions in the workbook to complete the lab
- Complete the online quiz for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module and select the quiz
- Provide feedback on the module
  - Log onto SKI, go to “My Learning” page
  - Find the module and select the “Leave Feedback” button to leave your comments

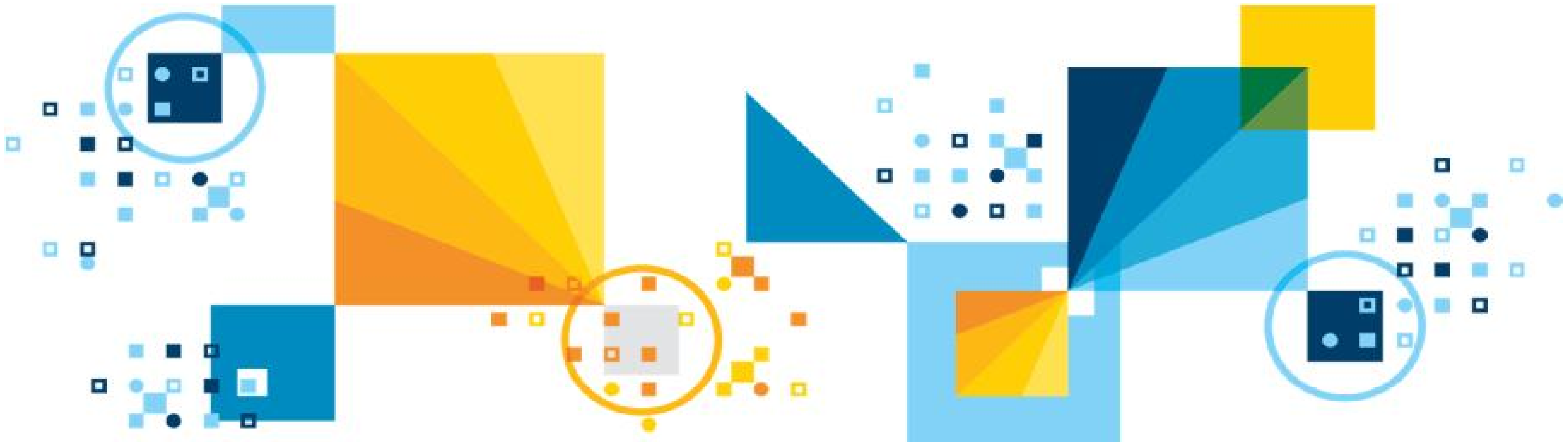
PRESENTATION NOTES FOR PAGE 79

The Next Steps…

§ The next set of modules to consider :
   – Module DB2 Performance Monitoring Essentials II

PRESENTATION NOTES FOR PAGE 80

Questions?
askdata@ca.ibm.com
Subject: DB2 10.5 Performance



PRESENTATION NOTES FOR PAGE 81