IBM DB2 10.5
for Linux, UNIX, and Windows

DB2 Workload Management Guide and Reference
Updated October, 2014
IBM DB2 10.5
for Linux, UNIX, and Windows

*DB2 Workload Management Guide and Reference*
*Updated October, 2014*
Contents

About this book .................................. ix

Chapter 1. Introduction to DB2 workload
management concepts .......................... 1
Domains of workload management .......... 1
Workload management administrator authority
(WLMADM) .................................. 3
Frequently asked questions about DB2 workload
management .................................. 4

Chapter 2. Work identification ................. 15
Activities ...................................... 15
Workload management DDL statements .... 18
Work identification by origin with workloads .. 19
  Workload assignment ........................ 23
  Default workloads .......................... 25
  Creating a workload ....................... 30
  Altering a workload ....................... 31
  Enabling a workload ....................... 33
  Disabling a workload ..................... 34
  Dropping a workload ...................... 34
  Permitting occurrences of a workload to access
the database ................................ 35
  Preventing occurrences of a workload from
accessing the database ...................... 36
  Granting the USAGE privilege on a workload .. 36
  Revoking the USAGE privilege on a workload .. 38
  Example: Workload assignment ............. 38
  Example: Workload assignment when workload
attributes have single values ............... 42
  Example: Workload assignment for a unit of
work when multiple workloads exist ....... 44
  Example: Workload assignment when workload
attributes have multiple values ............ 47
Work identification by type, cost, or data accessed
with work classes .......................... 48
  Work classes and work class sets .......... 51
  Evaluation order of work classes in a work class
set ........................................ 58
  Assignment of activities to work classes .. 59
  Work classifications supported by thresholds .. 60
  Example: Analyzing workloads by activity type 61
  Example: Using a work class set to manage
specific types of activities ................. 62
  Example: Working with a work class defined
with the ALL keyword ..................... 63

Chapter 3. Activities management .......... 67
Resource assignment with service classes .... 67
  Default service superclasses and subclasses .. 70
  Activity-to-service class mapping .......... 72
  Prefetch priority of service classes ........ 76
  Buffer pool priority of service classes .... 76
  States of connections and activities in a service
class ....................................... 77
  System-level entities not managed by service
classes .................................... 78
  Creating a service class ................... 79
  Altering a service class .................. 80
  Dropping a service class ................ 83
  Example: Using service classes .......... 84
  Example: Analyzing a service class-related
system slowdown .......................... 89
  Example: Investigating agent usage by service
class ..................................... 90
  Apply controls to types of activities with work
action sets ................................ 91
  How work classes, work class sets, work actions,
and work action sets work together and are
associated with other DB2 objects ......... 92
  Work actions and work action sets .......... 94
  Work actions and the work action set domain .... 104
  Thresholds that can be used in work actions .. 108
  Application of work actions to database
activities ................................ 108
  Concurrency control at the workload level using
work action sets .......................... 110
  Workload and work action set comparison .... 112
  Example: Using a database work action set and
database threshold ....................... 114
  Example: Using work action sets to determine
the types of work being run ............... 116
  Control of work with thresholds .......... 117
  Threshold domain and enforcement scope .... 120
  Threshold evaluation order ............... 121
  Creating a threshold ..................... 123
  Altering a threshold ..................... 124
  Dropping a threshold .................... 125
  Example: Using thresholds ............... 126
  Connection thresholds ................... 128
  Activity thresholds ..................... 128
  Aggregate thresholds .................... 137
  Unit of work thresholds ................. 144
  Priority aging of ongoing work .......... 145
  Sample priority aging scripts .......... 149
  Remapping activities between service subclasses .... 155
  Workload management dispatcher overview .. 158
  Workload management dispatcher .......... 159
  Hard CPU shares ........................ 178
  Soft CPU shares ........................ 185
  Enabling and setting CPU shares .......... 188
  CPU limit ................................ 190
  Minimum CPU resource utilization for service
class to be considered active ............ 205
  Dispatch concurrency level ............... 208
  Monitoring and tuning workload management
dispatcher performance .................... 210
  Default query concurrency management .... 213
  Default workload management objects for
concurrency control ..................... 217
  Canceling activities ..................... 222
MON_GET_WORK_ACTION_SET_STATS - Return work action set statistics .......... 414
MON_GET_ACTIVITY table function - Return a
list of activities ....... 415
MON_GET_WORKLOAD_STATS table function
- Return workload statistics .......... 432
WLM_SET_CLIENT_INFO procedure - Set client
information ............. 435
Workload management monitor elements .... 438
act_cpu_time_top - Activity CPU time top
monitor element .............. 438
act_exec_time - Activity execution time monitor
element ............... 438
act_remap_in - Activities remapped in
monitor element .............. 439
act_remap_out - Activities remapped out
monitor element .............. 439
act_rows_read_top - Activity rows read top
monitor element .............. 439
act_throughput - Activity throughput monitor
element .............. 440
act_total - Activities total monitor element .... 441
activate_timestamp - Activate timestamp
monitor element .............. 441
activity_collected - Activity collected monitor
element .............. 442
activity_id - Activity ID monitor element .... 442
activity_secondary_id - Activity secondary ID
monitor element .............. 443
activity_type - Activity type monitor element .. 443
agg_temp_tablespace_top - Aggregate
temporary table space top monitor element ... 444
arm_correlator - Application response
measurement correlator monitor element ... 444
bin_id - Histogram bin identifier monitor
element .............. 445
dset - Histogram bin bottom monitor element ... 445
current WTO_act_top - Concurrent WLO
activity top monitor element ........ 446
current WTO_act_top - Concurrent workload
occurrences top monitor element .......... 446
currentdbcoordactivities db_threshold_id -
Current database coordinator
activities database threshold ID monitor
element .............. 448
currentdbcoordactivities db_threshold
queued - Concurrent database coordinator
activities database threshold queued monitor
element .............. 448
currentdbcoordactivities db_threshold_value - Concurrent database
coordinator activities database threshold value
monitor element .............. 448
currentdbcoordactivities db_threshold_violated - Concurrent database
coordinator activities database threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_id - Concurrent database coordinator
activities service subclass threshold ID monitor
element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
currentdbcoordactivities subclass_threshold_value - Concurrent database coordinator
activities service subclass threshold
value monitor element .............. 449
currentdbcoordactivities subclass_threshold_violated - Concurrent database coordinator
activities service subclass threshold
violated monitor element .............. 449
WLM_SET_CLIENT_INFO procedure - Set client
information ............. 435
Contents V
<table>
<thead>
<tr>
<th>Monitor Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord_act_lifetime_top</td>
<td>Coordinator activity lifetime top monitor element</td>
</tr>
<tr>
<td>coord_act_queue_time_avg</td>
<td>Coordinator activity queue time average monitor element</td>
</tr>
<tr>
<td>coord_act_rejected_total</td>
<td>Coordinator activities rejected total monitor element</td>
</tr>
<tr>
<td>coord_partition_num</td>
<td>Coordinator partition number monitor element</td>
</tr>
<tr>
<td>cost_estimate_top</td>
<td>Cost estimate top monitor element</td>
</tr>
<tr>
<td>cpu_limit</td>
<td>WLM dispatcher CPU limit monitor element</td>
</tr>
<tr>
<td>cpu_share_type</td>
<td>WLM dispatcher CPU share type monitor element</td>
</tr>
<tr>
<td>cpu_shares</td>
<td>WLM dispatcher CPU shares monitor element</td>
</tr>
<tr>
<td>cpu_utilization</td>
<td>CPU utilization monitor element</td>
</tr>
<tr>
<td>cpu_velocity</td>
<td>CPU velocity monitor element</td>
</tr>
<tr>
<td>db_work_action_set_id</td>
<td>Database work action set ID monitor element</td>
</tr>
<tr>
<td>db_work_class_id</td>
<td>Database work class ID monitor element</td>
</tr>
<tr>
<td>destination_service_class_id</td>
<td>Destination service class ID monitor element</td>
</tr>
<tr>
<td>estimated_cpu_entitlement</td>
<td>Estimated CPU entitlement monitor element</td>
</tr>
<tr>
<td>histogram_type</td>
<td>Histogram type monitor element</td>
</tr>
<tr>
<td>last_wlm_reset</td>
<td>Time of last reset monitor element</td>
</tr>
<tr>
<td>num_remaps</td>
<td>Number of remaps monitor element</td>
</tr>
<tr>
<td>num_threshold_violations</td>
<td>Number of threshold violations monitor element</td>
</tr>
<tr>
<td>number_in_bin</td>
<td>Number in bin monitor element</td>
</tr>
<tr>
<td>parent_activity_id</td>
<td>Parent activity ID monitor element</td>
</tr>
<tr>
<td>parent_uow_id</td>
<td>Parent unit of work ID monitor element</td>
</tr>
<tr>
<td>prep_time</td>
<td>Preparation time monitor element</td>
</tr>
<tr>
<td>queue_assignments_total</td>
<td>Queue assignments total monitor element</td>
</tr>
<tr>
<td>queue_size_top</td>
<td>Queue size top monitor element</td>
</tr>
<tr>
<td>queue_time_total</td>
<td>Queue time total monitor element</td>
</tr>
<tr>
<td>request_exec_time_avg</td>
<td>Request execution time average monitor element</td>
</tr>
<tr>
<td>routine_id</td>
<td>Routine ID monitor element</td>
</tr>
<tr>
<td>rows_fetched</td>
<td>Rows fetched monitor element</td>
</tr>
<tr>
<td>rows_modified</td>
<td>Rows modified monitor element</td>
</tr>
<tr>
<td>rows_returned</td>
<td>Rows returned monitor element</td>
</tr>
<tr>
<td>sc_work_action_set_id</td>
<td>Service class work action set ID monitor element</td>
</tr>
<tr>
<td>sc_work_class_id</td>
<td>Service class work class ID monitor element</td>
</tr>
<tr>
<td>section_env</td>
<td>Section environment monitor element</td>
</tr>
<tr>
<td>service_class_id</td>
<td>Service class ID monitor element</td>
</tr>
<tr>
<td>service_subclass_name</td>
<td>Service subclass name monitor element</td>
</tr>
<tr>
<td>service_superclass_name</td>
<td>Service superclass name monitor element</td>
</tr>
<tr>
<td>source_service_class_id</td>
<td>Source service class ID monitor element</td>
</tr>
<tr>
<td>statistics_timestamp</td>
<td>Statistics timestamp monitor element</td>
</tr>
<tr>
<td>stmt_invocation_id</td>
<td>Statement invocation identifier monitor element</td>
</tr>
<tr>
<td>temp_tablespace_top</td>
<td>Temporary table space top monitor element</td>
</tr>
<tr>
<td>thresh_violations</td>
<td>Number of threshold violations monitor element</td>
</tr>
<tr>
<td>threshold_action</td>
<td>Threshold action monitor element</td>
</tr>
<tr>
<td>threshold_domain</td>
<td>Threshold domain monitor element</td>
</tr>
<tr>
<td>threshold_maxvalue</td>
<td>Threshold maximum value monitor element</td>
</tr>
<tr>
<td>threshold_name</td>
<td>Threshold name monitor element</td>
</tr>
<tr>
<td>threshold_predicate</td>
<td>Threshold predicate monitor element</td>
</tr>
<tr>
<td>threshold_queuesize</td>
<td>Threshold queue size monitor element</td>
</tr>
<tr>
<td>thresholdid</td>
<td>Threshold ID monitor element</td>
</tr>
<tr>
<td>time_completed</td>
<td>Time completed monitor element</td>
</tr>
<tr>
<td>time_created</td>
<td>Time created monitor element</td>
</tr>
<tr>
<td>time_ofViolation</td>
<td>Time of violation monitor element</td>
</tr>
<tr>
<td>time_started</td>
<td>Time started monitor element</td>
</tr>
<tr>
<td>top</td>
<td>Histogram bin top monitor element</td>
</tr>
<tr>
<td>total_disp_run_queue_time</td>
<td>Total dispatcher run queue time monitor element</td>
</tr>
<tr>
<td>uow_completed_total</td>
<td>Total completed units of work monitor element</td>
</tr>
<tr>
<td>uow_comp_status</td>
<td>Unit of Work Completion Status</td>
</tr>
<tr>
<td>uow_elapsed_time</td>
<td>Most Recent Unit of Work Elapsed Time</td>
</tr>
<tr>
<td>uow_id</td>
<td>Unit of work ID monitor element</td>
</tr>
<tr>
<td>uow_lifetime_avg</td>
<td>Unit of work lifetime average monitor element</td>
</tr>
<tr>
<td>uow_lock_wait_time</td>
<td>Total time unit of work waited on locks monitor element</td>
</tr>
<tr>
<td>uow_log_space_used</td>
<td>Unit of work log space used monitor element</td>
</tr>
<tr>
<td>uow_start_time</td>
<td>Unit of work start timestamp monitor element</td>
</tr>
<tr>
<td>uow_status</td>
<td>Unit of Work Status</td>
</tr>
<tr>
<td>uow_stop_time</td>
<td>Unit of work stop timestamp monitor element</td>
</tr>
<tr>
<td>uow_throughput</td>
<td>Unit of work throughput monitor element</td>
</tr>
</tbody>
</table>
About this book

This book provides information on the DB2® workload management features and functionality that can help you obtain a stable, predictable execution environment that meets your business objectives. Using DB2 workload management, both requests and resources are managed. This book also provides information on monitoring and performing troubleshooting for the workload on your data server.
Chapter 1. Introduction to DB2 workload management concepts

A good workload management system helps to efficiently meet goals in the environment where work occurs. You can see examples of the need for a good workload management system all around you.

For example, look at a grocery store. Different activities must be considered: serving customers, stocking shelves, maintaining inventories, and so on. And some simple goals must be set. Store owners want to maximize both the number of customers who move through the store, and the amount that customers purchase, achieving both goals in a way that customers leave both satisfied and wanting to come back. Store owners must also ensure that they have sufficient stock for their customers to buy (but not too much stock, because waste becomes an issue). Store owners also track what their customers purchase, and use this information to create advertisements that are designed to induce their customers to return. Monitoring mechanisms track inventory and send notifications when stocks run low. Security devices are in place to detect shoplifting. Special fast checkout lanes are created so that shoppers who only want to purchase a few items can do so without having to wait behind other customers who are purchasing many items. If all of these goals are met and all of these operational procedures work well, customers are satisfied, and are likely to return rather than to go to another store. These goals and operational procedures are all aspects of workload management.

In a data server environment, you can see even more of a need for effective management of work, especially now that data servers are being stressed like never before. Cash registers generate thousands of data inserts, reports are constantly being generated to determine whether sales targets are being met, batch applications run to load collected data, and administration tasks such as backups and reorganizations run to protect the data and make the server run optimally. All these operations are using the same database system and competing for the same resources.

To ensure the best chance of meeting goals for running a data server, an efficient workload management system is critical.

Domains of workload management

Workload management has three clearly defined domains: identification of the work entering the data server, management of the work when it is running, and monitoring to ensure that the data server is being used efficiently.

A number of aspects must be considered for successful workload management with DB2 workload manager, starting with understanding your goals. In the grocery store example described in Chapter 1, “Introduction to DB2 workload management concepts,” goals might include maximizing customer spending, minimizing shoplifting, and ensuring that customers leave the store satisfied so that they will return again.

In a data server environment, you must also define goals. Sometimes the goals are clear, especially when they originate from service level agreement (SLA) objectives. For example, queries from a particular application can consume no more than 10%
of the total processor resource. Goals can also be tied to a particular time of day. For example, an overnight batch utility might have to complete loading data by 8 a.m. so that the daily sales reports are on time. In other situations, the goals can be difficult to quantify. A goal might be to keep the database users satisfied and to prevent aberrant database activity from hampering their day-to-day work. Whether the goals are quantifiable or not, understanding them is critical when considering the following stages of workload management:

**Identification**

If you want to achieve a goal for some kind of work, you first must be able to identify details about the work. In the grocery store, you can identify shopper information through credit cards and debit cards, or an unpaid-for item through an active security tag on the item. For the data server, you need to decide how you want to identify the work that enters the system. You can use the name of the application that submits the work, the authorization ID that submits the work, or a combination of elements that provide some form of identification.

**Management**

The management phase includes mechanisms for making steady progress towards your goal, and actions to take if a goal is not being met. An example of a mechanism is managing price checks in fast checkout lanes. Fast checkout lanes should result in faster throughput and satisfied customers, but if a carton of milk has the wrong price and a price check is required, the fast checkout lane could slow down. The problem is managed by performing a fast price check, possibly opening up another checkout lane, and trying to fix the pricing problem so that it does not occur again. On the data server, you might find that overall performance is suffering when a few poorly written SQL statements are running, a surge in volume occurs during peak times, or there is too much competition between different applications for the same resources. The management phase includes mechanisms for assigning resources to achieve your goals and actions to take if a goal is not being met. These workload management mechanisms, which indirectly and directly control CPU resources, include the following:

1. Concurrency thresholds, applied with a work action set defined on the workload, to control the concurrency of incoming work
2. The ability to move work form one service class to another, currently only effective when workload management dispatcher is being used on those service classes
3. Workload management dispatcher to specifically allocate CPU resources for workloads assigned to service classes for more granular control of CPU resources when the first two workload management mechanisms are not adequate

**Monitoring**

Monitoring is important for a couple of reasons. First, to determine whether you are achieving a goal, you must have a mechanism to track progress toward that goal. Also, monitoring helps to identify the problems that might be preventing you from achieving your goal. In a store, the store manager can watch the flow of customers, automatically be alerted to problems such as shoplifting or dangerously low inventory of a particular sale item, or perform analysis on historical purchase patterns to determine optimal product placement in the store. For a data server, there are often explicit goals for response times of database activities and it is important to have a way to measure this metric, and watch for trends.
The following figure represents the workload management phases:

![Workload management phases diagram](image.png)

**Figure 1. Phases of workload management**

---

**Workload management administrator authority (WLMADM)**

You need workload management administrator (WLMADM) authority to manage workload objects for a specific database. This authority enables you to create, alter, drop, comment on, grant access to, and revoke access from DB2 workload management objects.

Workload management objects are system objects similar to buffer pools and table spaces. As such, they do not have owners associated with them.

The security administrator, who is someone holding SECADM authority, or a user with ACCESSCTRL authority can grant WLMADM authority to a user, group, or role.

WLMADM authority provides the ability to perform the following tasks:

- Issue CREATE, ALTER, COMMENT ON, and DROP statements for the following DB2 workload management objects:
  - Histogram templates
  - Service classes
  - Thresholds
  - Work action sets
  - Work class sets
  - Workloads
- Issue GRANT and REVOKE statements for workload privileges

If you hold database administrator (DBADM) authority, you hold WLMADM authority implicitly.
Frequently asked questions about DB2 workload management

This FAQ provides you with answers to common questions about DB2 workload management.

**General**
- On which DB2 platforms can I use DB2 workload management?
- I am not on AIX®. Does this mean I do not have any control over processor resource or I/O activity?
- Now that Query Patroller is discontinued and DB2 Governor are deprecated, how do I migrate to DB2 workload manager?
- Is there a way for WebSphere® Application Server to pass the client information fields used by the DB2 workload?
- Why is my work not assigned to the correct workload?
- Why does DB2 workload manager affect REORGCHK, IMPORT, EXPORT and other CLP commands?
- Is there a way to change the service class to which an activity is assigned while it is executing?
- Much of my batch work is done using CLP scripts under the same ID, how can I go about uniquely identifying these so I can manage them differently from each other?
- When should I use the COLLECT AGGREGATE ACTIVITY DATA clause versus the COLLECT ACTIVITY DATA clause?
- How does DB2 workload management work with the new AIX WPAR feature?
- What is the relationship between the DB2_OPT_MAX_TEMP_SIZE registry variable and a DB2 threshold based on SQLTEMPSPACE?

**Licensing**
- What are the licensing requirements for DB2 workload manager?

**Monitoring**
- What information do you get from the different event monitors that are associated with workload management?

**OS workload management (AIX WLM and Linux WLM)**
- Why would I ever want to use AIX WLM or Linux WLM?
- I am not on AIX. Does this mean I do not have any control over processor resources or I/O activity?
- Can I use AIX WLM to manage I/O activity?
- Can I use AIX WLM to manage memory use?
- How does DB2 WLM work with the new AIX WPAR feature?

**Platforms**
- On which DB2 platforms can I use DB2 workload manager?
- I am not on AIX. Does this mean I do not have any control over processor resource or I/O activity?
- How does DB2 workload management work with the new AIX WPAR feature?
- Why would I ever want to use AIX WLM or Linux WLM?
• How does this new functionality affect Query Patroller and DB2 Governor?
• Now that Query Patroller and DB2 Governor are deprecated, how do I migrate to DB2 workload manager?

Thresholds
• Can I create multiple CONCURRENTDBCOORDACTIVITIES concurrency thresholds for the same set of work?
• How do I determine which activities are queued by a workload management threshold and the order of the activities in the queue?

Workload management dispatcher
• Do I need to use workload management dispatcher?
• What changes in behavior might I see when I turn ON the workload management dispatcher?
• With the introduction of the workload management dispatcher, are concurrency thresholds such as CONCURRENTDBCOORDACTIVITIES no longer needed or useful?

On which DB2 platforms can I use DB2 workload management?

DB2 workload management is available on all platforms supported by DB2 9.5 for Linux, UNIX, and Windows or later. The optional tight integration-offered between DB2 workload management at the service class level and operating system workload management capabilities—is available on AIX platforms and any Linux platform based on the 2.6.26 kernel or higher.

Do I need to use workload management dispatcher?

Most workload management configurations begin with concurrency thresholds, which affect the consumption of all resources by controlling how much work can begin executing at any one time. In some situations, however, a concurrency threshold is not able to effectively limit the total amount of processing resource that is consumed, and high priority work is affected; for example, a scenario in which complex work is restricted to one running query that nevertheless consumes enough resource to disrupt higher priority work. In such cases, the workload management dispatcher is used to explicitly control CPU consumption and protect the higher priority work.

You can use the workload management dispatcher for any or all of the following situations:
• You want to manage the share of CPU resources among multiple users or applications and you are using an operating system that does not have an operating system (OS) workload manager that integrates with DB2 workload management through the outbound_correlator field on each service class.
• You want to manage the share of CPU resources among multiple users or applications and you do not have root privilege on the operating system.
• You want to manage the share of CPU resources among multiple users or applications in a multiple member environment across multiple systems and managing this through the OS WLM on each system requires too much administration.
You want to manage the share of CPU resources among multiple users or applications using hard shares to limit certain service classes, even when the CPU is under-utilized, and this is not available in your OS WLM or does not produce the desired result.

**How does this new functionality affect Query Patroller and DB2 Governor?**

The DB2 workload manager introduces an independent approach to workload management and does not rely on or interact with Query Patroller or DB2 Governor in any way. Query Patroller has been discontinued starting with the Version 10.1 release. DB2 Governor was deprecated in the DB2 Version 9.7 release and, although still functional, it is no longer central to the DB2 workload management strategy. No further investment is planned for DB2 Governor in future releases.

When DB2 9.5 or later is first installed, the default user service class is automatically defined and all incoming work is sent to it for execution. Although DB2 Governor can watch agents in any service class, it is permitted to adjust the agent priority only for agents in the default user service class.

**I am not on AIX. Does this mean I do not have any control over processor resources or I/O activity?**

Users on all platforms have the same ability to control processor resources and I/O activity between service classes using SQL, such as the CREATE and ALTER SERVICE CLASS statements, for example.

To control CPU usage when the workload management dispatcher is enabled, use the CPU limit attribute of the DB2 service class to limit the amount of CPU resources a service class can consume. If the workload management dispatcher CPU shares (wlm_disp_cpu_shares) database manager configuration parameter is also enabled, you can use the CPU shares attribute of the DB2 service class to specify the share of CPU resources that a service class can consume relative to the CPU consumption of other service classes. On AIX and some Linux platforms, you can supplement (or replace) these approaches by taking advantage of the workload management capabilities that are offered by those operating systems to control CPU consumption.

For I/O activity, users on all platforms can set the buffer pool or prefetcher priority attribute of a DB2 service class to a value of high, medium, or low. All service classes run with a medium priority by default.

**Can I use AIX or Linux WLM or the DB2 workload management dispatcher to manage I/O activity?**

Currently, neither AIX WLM nor Linux WLM support direct I/O activity controls at the thread level. However, it is possible to indirectly control I/O activity by means of concurrency thresholds, or to use the DB2 workload management dispatcher, AIX WLM, or Linux WLM to manipulate CPU resources. The more CPU resource that is available to an executing thread, the less frequently that thread will request I/O resources.

You can influence buffer pool behavior by using the BUFFERPOOL PRIORITY attribute of any DB2 service class. You can also control DB2 prefetcher I/O activity by using the PREFETCH PRIORITY attribute of any DB2 service class.
Can I use AIX or Linux WLM to manage memory use?

DB2 data server uses primarily shared memory, which is accessed by more than one agent from different service classes. For this reason, it is not possible to divide memory allocation between different service classes using either AIX or Linux WLM.

Memory (such as sortheap) that is consumed during the execution of an SQL statement can be indirectly influenced through the use of concurrency thresholds, because consumption does not begin until the statement is allowed to execute. However, unlike I/O activity, restricting CPU consumption does not affect the amount of memory that is consumed. In fact, restricting CPU consumption can exacerbate the memory situation, because queries will be running more slowly and holding onto their allocated memory longer.

Is there a way for WebSphere Application Server to pass the client information fields used by the DB2 workload?

WebSphere Application Server Version 6.0 and Version 6.1 can set or pass in the CLIENT INFO fields to DB2 data server, either explicitly by your applications (see: Passing client information to a database) or implicitly by having WebSphere Application Server do it for you (see: Implicitly set client information).

Can I create multiple CONCURRENTDBCOORDACTIVITIES concurrency thresholds for the same set of work?

You can create one or more CONCURRENTDBCOORDACTIVITIES concurrency thresholds that apply to the same set of activities by defining them at the level of the database, the service class in which the work executes, or within a work action set applied at the database or workload level. Be aware that each new concurrency threshold that applies to an activity implies additional overhead to enforce that concurrency threshold. Verify that you really need more than one concurrency threshold level.

Why is my work not assigned to the correct workload?

There are a number of reasons why a connection may not be mapped to the desired workload. The most common ones are the failure to grant USAGE privilege on the workload, incorrect spelling of the case sensitive connection attributes, or the existence of a matching workload definition that is positioned earlier in the evaluation order.

Before a connection can be assigned to a workload, the connection attributes must match those of the workload definition, and the session authorization ID must have USAGE privilege on the workload. A common omission is to create the workload but not to grant USAGE privilege on the workload to users (See “GRANT (Workload Privileges) statement” in SQL Reference). Only users with ACCESSCTRL, SECADM, or WLMADM authority can grant workload usage privilege to other users. Users with ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority have implicit usage privilege on all workloads.

Connection attributes for workloads are case sensitive. For example: If the system user ID is uppercased, then the SYSTEM_USER connection attribute you specify must be in uppercase as well.
To establish why a connection is not being mapped to the expected workload, you should gather some information. Which workload is the work being mapped to? Is that workload before or after the one that you thought would be used when you look at the workload definitions in the order of evaluation? (Hint: try selecting the workload definitions ordered in ascending order by the value of the EVALUATIONORDER column in SYSCAT.WORKLOADS).

If you do not know what the connection attributes are for the target connection, you can find out the values for the connection in a number of different ways:

- Issue a query against the system using the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function while the connection is active
- Open a cursor on a connection and use the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure against that cursor to have the activity information captured to the activities event monitor (Hint: do not forget to create and activate the activities information event monitor)
- Turn on the collection of detailed activity information for the workload being used by the connection, issue one statement in order to capture the activity information, and then turn off the collection.

**Why does DB2 workload manager affect REORGCHK, IMPORT, EXPORT and other CLP commands?**

These CLP commands are affected by DB2 workload management thresholds, because the database engine cannot distinguish system requests originating with these utilities from other requests directly initiated by users within the CLP interactive front-end.

**Is there a way to change the service class to which an activity is assigned while it is executing?**

Yes, you can change the service subclass an activity is executing in to another service subclass within the same parent service superclass by defining a CPUTIMEINSC, DATATAGINSC, or SQLROWSINSC threshold with the REMAP ACTIVITY action on the original service subclass. Initially, DB2 workload management maps an activity to a service class based on the relevant workload definition for the connection, modifies it as required if a work action set exists on that service class, and then sets up the DB2 agent to execute in the assigned service class. When an activity violates a threshold that has a REMAP ACTIVITY action defined, the agent remaps itself to the specified target service subclass (under the same superclass) once the threshold violation has been detected and the activity continues executing in the new service subclass.

**Much of my batch work is done using CLP scripts under the same ID, how can I go about uniquely identifying these so I can manage them differently?**

You have a couple of options:

An enhancement has been added to CLP so that the client application name is automatically set to the CLP script filename, with a CLP prefix preceding it (the value of this field at the server can be seen in the CURRENT CLIENT_APPLNAME special register). For example, if the CLP script filename is batch.db2, the CURRENT CLIENT_APPLNAME special register value is set to CLP
**batch.db2** by CLP when that script is run. With this feature, it is possible for different CLP scripts to be associated with different workloads based on the client application name.

For example, to create a workload for CLP file batch1.db2, you can issue the following DDL statement:

```sql
CREATE WORKLOAD batch1 CURRENT CLIENT_APPLNAME ('CLP batch1.db2')
SERVICE CLASS class1
```

To create a workload for CLP file **batch2.db2**, you can issue the following DDL statement:

```sql
CREATE WORKLOAD batch2 CURRENT CLIENT_APPLNAME ('CLP batch2.db2')
SERVICE CLASS class2
```

Since these two batch files are associated with different workloads, they can be assigned to different service classes and managed differently.

Another option is the new stored procedure **WLM_SET_CLIENT_INFO**, which permits you to set the values of any of the client information fields at the server using a simple CALL SQL statement. By inserting a CALL statement into any of your existing CLP scripts, you can uniquely identify them using these fields and map them to different workload definitions.

For more information, see “WLM_SET_CLIENT_INFO procedure” in *Administrative Routines and Views*.

**When should I use the COLLECT AGGREGATE ACTIVITY DATA clause versus the COLLECT ACTIVITY DATA clause?**

The answer depends on why the monitoring is desired and what is to be done with the information.

Aggregate activity information spans the entire set of work that has executed within the scope covered by the clause, and it captures summary characteristics of this set; it does not capture specific details about individual activities. The COLLECT AGGREGATE ACTIVITY DATA clause can be specified on DB2 workloads, DB2 service classes, and DB2 work action sets. For normal operational monitoring, use the COLLECT AGGREGATE ACTIVITY DATA clause, because it is very lightweight, it can be gathered automatically by the statistics event monitor for a historical record, and it provides important information on overall response time patterns. If further insight is required on a specific type of work, use the COUNT ACTIVITY or COLLECT AGGREGATE ACTIVITY DATA actions within a DB2 work action set to gather more granular information (with minimal overhead) about different types of work executing in a workload, service class, or database.

In contrast, activity information contains detailed information about each and every activity that executes within the scope covered by the COLLECT ACTIVITY DATA clause. This clause can be specified on DB2 workloads, DB2 service classes, DB2 work action sets, and DB2 thresholds. It permits further in-depth analysis of the individual activities that are captured, in order to understand the flow and type of SQL statements submitted by a new application, for example, or to look into performance tuning opportunities with tools such as the Explain facility or the Design Advisor. Because it captures much more information for each activity affected by it, the impact of using this clause is higher on affected activities than other monitoring methods and it should be carefully controlled.
How does DB2 workload management work with the new AIX WPAR feature?

All aspects of DB2 workload management will work within an AIX WPAR but because AIX WPARs do not support the use of AIX WLM features, the option to tightly integrate DB2 service classes with AIX WLM service classes is of no benefit in this environment.

What is the relationship between the DB2_OPT_MAX_TEMP_SIZE registry variable and DB2 thresholds based on SQLTEMPSPACE?

There is no direct relationship between these two things. The DB2_OPT_MAX_TEMP_SIZE registry variable is a directive to the query compiler to limit the amount of temporary table space that a query can use. This can cause the optimizer to choose a plan that is more expensive (potentially less efficient) but which uses less space in the system temporary table spaces. A DB2 threshold based on SQLTEMPSPACE does not affect the type of plan chosen by the optimizer. It simply causes DB2 data server to monitor the usage of system temporary table space by that query at each member and generates a threshold violation if the stated limit is exceeded during normal processing.

Now that Query Patroller is discontinued and DB2 Governor is deprecated, how do I migrate to DB2 workload manager?

Following the introduction of DB2 workload manager as the strategic workload management solution in DB2 Version 9.5, Query Patroller has been discontinued in the Version 10.1 release and the DB2 Governor has been deprecated since the DB2 Version 9.7 release and might be removed in a future release.

Although DB2 Governor is still supported in this release, you should begin adopting the new features and capabilities of DB2 workload manager, including those introduced in this release. Note that with DB2 workload manager, you have many more options, and you should explore them, which might require you to rethink your approach to controlling work on your DB2 data server in current workload management terms. The DB2 best practices article Implementing DB2 workload management in a data warehouse contains a supplement that is specifically designed for those who are migrating from Query Patroller. Pertinent task topics are also available in the Related tasks section.

To facilitate migration from DB2 Query Patroller to DB2 workload manager, a sample script (qpwlmmig.pl) has been included starting with DB2 V9.7 Fix Pack 1. For additional information, see one of the following tasks for details on how to migrate from Query Patroller to DB2 workload manager:

- Migrating from Query Patroller to DB2 workload manager using the sample script
- Migrating from Query Patroller to DB2 workload manager

What are the licensing requirements for DB2 workload manager?

A subset of the workload management capabilities in DB2 data server has its use restricted through licensing. This licensed subset is referred to as DB2 workload manager, and it controls the creation of any service class, workload, threshold, or work action set. Access to this subset of workload management capabilities requires one of the following licensed products:
IBM® DB2 10.5 Enterprise Server Edition
IBM DB2 10.5 Advanced Enterprise Server Edition
IBM Developer Edition
IBM Smart Analytics System

The following workload management functions are not restricted by license:

- Using or altering the default service classes and workloads; this includes all monitoring capabilities
- Creating, altering, or dropping histogram templates
- Using the DB2 workload management table functions or stored procedures
- Creating, activating, stopping, or dropping workload management event monitors
- Granting, altering, or revoking workload privileges

**What information do you get from the different event monitors that are associated with workload management?**

The threshold violations, statistics, and activities event monitors capture information about threshold violations, operational statistics and aggregate activity data, and individual activity data (see: “Historical monitoring with WLM event monitors” on page 237).

Each event monitor collects one or more logical data groups (see: “Event type mappings to logical data groups” in Database Monitoring Guide and Reference) and there are one or more monitoring elements in each logical data group (see: “Event monitor logical data groups and monitor elements” in Database Monitoring Guide and Reference).

For example, to discover what information is collected by the threshold violations event monitor, start by looking in Table 3 in “Event type mappings to logical data groups” topic. This table shows that the threshold violations event monitor collects information into a single logical data group called event_thresholdviolations (note that some event monitors, like the activity event monitor, collect information into multiple logical data groups). Next, find the event_thresholdviolations logical data group in “Event monitor logical data groups and monitor elements” topic. This topic shows which monitor elements are reported in the event_thresholdviolations logical data group, which includes the following:

- **activate_timestamp** - Activate timestamp
- **activity_collected** - Activity collected
- **activity_id** - Activity ID
- **agent_id** - Application Handle (agent ID)
- **appl_id** - Application ID
- **coord_partition_num** - Coordinator partition number
- **destination_service_class_id** - Destination service class ID
- **source_service_class_id** - Source service class ID
- **threshold_action** - Threshold action
- **threshold_maxvalue** - Threshold maximum value
- **threshold_predicate** - Threshold predicate
- **threshold_queuesize** - Threshold queue size
- **thresholdid** - Threshold ID

Chapter 1. Introduction to DB2 workload management concepts
The approach outlined in this example can be used to discover what data is collected by each event monitor.

How do I determine which activities are queued by a workload management threshold and the order of the activities in the queue?

You can do this by first creating a view using the WLM_GET_SERVICE_CLASS_AGENTS table function and then running statements to list the queued activities in the order of the queue entry time. For examples describing how to do this, see: “Example: Determining which activities are queued by a WLM threshold and their queue order” on page 235.

What changes in behavior might I see when I turn ON the workload management dispatcher?

If you enable CPU shares via the wlm_disp_cpu_shares database manager configuration parameter and do not specify CPU shares or CPU limits for your service classes, all service classes receive an equal soft share of the CPU resources on your system. The effect of all service classes receiving an equal soft share of the CPU resources might result in a different allocation of CPU resources to services classes than in previous DB2 releases. As a result, you should consider setting CPU shares or CPU limit values appropriate for your workload. For more information about how to determine values for CPU shares and CPU limits, see: “Workload management dispatcher” on page 159.

With the introduction of the workload management dispatcher, are concurrency thresholds such as CONCURRENTDBCOORDACTIVITIES no longer needed or useful?

The DB2 workload management dispatcher and concurrency thresholds can be used together. Concurrency thresholds are still very useful for controlling how much work is running. For each activity that starts running, the DB2 database manager provides other resources to that activity, in addition to CPU resources, which the activity usually retains for as long as it is running. Such non-CPU resources include (among others) the DB2 agent, sort memory, temporary table space, locks, and I/O. By preventing an activity from starting to run, those additional non-CPU resources are not consumed and are available for other activities.

In addition, concurrency thresholds can be applied at different points within the DB2 database manager to determine the origin of the work that is running. For example, putting a concurrency threshold on large queries coming from a specific workload limits the consumption or share of the resources available to that particular workload in a service class, as compared to other workloads contributing to the same service class.

In summary, concurrency thresholds can be used to control when activities start to run and consume the CPU and non-CPU resources on the system. The workload management dispatcher can be used to control how much of the CPU resources such activities get to consume once they start running.
Why would I ever want to use AIX WLM or Linux WLM?

Even if you use the DB2 workload management dispatcher for controlling the CPU consumption of your DB2 workloads, the following are reasons to use AIX WLM or Linux WLM as well:

- Operating system (OS) workload managers provide monitoring of resource consumption at the level of the operating system.
- OS workload managers can provide control for all processes or threads on the entire host or LPAR, not just DB2 database manager threads. This can help when there is a need to control processes that compete for resources with DB2 database manager.
Chapter 2. Work identification

A key part of successfully implementing a DB2 workload management solution is identifying the work.

There are three approaches to work identification: workloads, work classes, and data tags.

- Workloads can be used to identify the source of work. You can identify the source at the connection or transaction level by using key session attributes, such as the application name or the system authorization ID that submitted the work, for example.

- A work class enables you to identify work by pinpointing certain characteristics of the work. You can define a work class to identify the work of interest, such as statements that only modify data on the data server, for example (INSERT, UPDATE, or DELETE statements, for example).

- Data tags are available as table space and storage group data tags. They enable you to identify work indirectly by the type of data it accesses during execution. By tagging data of interest, you can identify the work that is accessing that data.

Activities

One way that you can monitor and control workloads is on the basis of individual activities. Each time your DB2 data server executes the access plan for an SQL or XQuery statement or executes the load utility a corresponding activity is created.

For workload monitoring, commonly used monitor elements provide information in terms of activity units. For example, you can obtain information about the workload volume and response time from monitor elements such as the activity execution time monitor element (coord_act_exec_time) and the high watermark for the concurrent activities (concurrent_act_top).

For workload control, most workload controls and thresholds apply to each activity. For example, the ACTIVITYTOTALTIME threshold controls the maximum time that your data server can spend processing an activity.

Statements or commands that trigger activities on your data server

The following statements or commands trigger activities on your data server:

- All DML statements
- All DDL statements
- The CALL statement
- The load utility

The life cycle of activities

The life cycle of an activity for a DML statement does not include processing that occurs before or outside of access plan execution. This implies that activity-based monitoring does not cover operations such as connecting to the database or compiling SQL into an access plan.
During its life cycle, an activity can spend time in various states, which are reported by the activity_state event monitor element. Some of the states an activity can be in are:

- **EXECUTING** - This state indicates that the coordinator agent is working on the activity. An activity that encounters a lock wait situation is reported as executing. The total amount of time spent executing activities is returned by the `total_act_time` monitor element.
- **IDLE** - This state indicates that the coordinator agent is waiting for the next request from a client.
- **QUEUED** - Some thresholds include a built-in queue. This state indicates that the activity is waiting in the queue for its turn to begin executing.

Monitoring data for the activity is aggregated at the end of the lifetime of an activity.

The following figure shows how the lifetime of a long running query breaks down into queue time and execution time:

![Diagram showing queue time and execution time]

**Types of SQL statements and application development**

This section describes what activities are created for various SQL statements and identifies the start and end points in the lifetime of these activities. You can use this information to understand how SQL statements are monitored and controlled through activities.

**SELECT statements**: A SELECT statement is represented by one activity. This includes any cursor requests such as FETCH operations and subselects or subqueries. The activity starts when your DB2 data server starts processing an OPEN cursor statement or request, and ends when your data server has completed processing for a CLOSE cursor statement or request.

**SELECT statements using WITH HOLD cursors**: When a WITH HOLD cursor is used, an application can open a cursor within one unit of work and close the cursor in a subsequent unit of work. The cursor remains open for multiple units of work. The corresponding activity exists for as long as the cursor is open, because the life cycle of the activity ends only after the cursor is closed.

**CALL statement and stored procedures**: A CALL statement itself is represented by one activity, but the payload of the stored procedure can spawn nested activities as follows:

<table>
<thead>
<tr>
<th>Contents of stored procedure</th>
<th>Additional activities created</th>
</tr>
</thead>
<tbody>
<tr>
<td>A single SQL statement</td>
<td>One</td>
</tr>
<tr>
<td>No SQL statements in the stored procedure</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 1. Contents of stored procedures and activities they create (continued)

<table>
<thead>
<tr>
<th>Contents of stored procedure</th>
<th>Additional activities created</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL procedures, multiple SQL statements, and looping logic</td>
<td>Multiple activities, one corresponding to each invocation of each statement</td>
</tr>
<tr>
<td>A call to another stored procedure</td>
<td>Activities for that stored procedure</td>
</tr>
</tbody>
</table>

The activity associated with the CALL statement starts when your DB2 data server starts processing the statement or request and ends after the stored procedure processing is complete.

Triggers and UDFs: When a SQL statement calls a trigger or UDF, no additional activity is created. The work done by that trigger or UDF is accrued to the activity for the SQL statement that called it. Cases where the trigger or UDF executes additional SQL statements are handled like any statement execution, that is, an activity is created for each statement.

PREPARE statement: No activity is created, because activities are not created until an access plan is executed.

**Nested activities**

Nested activities do not significantly affect activity-based monitoring and control of workloads, but some additional information applies.

Activities that can have nested activities within them are:

- A stored procedure
- An anonymous block
- An autonomous routine
- A DML activity that executes a UDF
- A load from cursor (a load activity that has the cursor activity nested within it)
- A DML activity that is subject to a trigger that contains any of the activities listed previously as part of the trigger definition

Nested activities are reported in monitoring information as follows:

- A nested activity is indicated by a nonzero parent UOW ID and a nonzero parent activity ID.
- A nested activity is not counted towards histograms or any statistics derived from histograms.
- Data for a nested activity is not also reported as part of the metrics for the parent activity. For example, if a procedure executed by a CALL statement performs an insert which consumes 10 seconds of processor time, that processor time is counted only towards the processor time metric for the insert activity and does not count towards the processor time metric for the parent CALL activity.

Workload control considers nested activities as follows:

- An activity nested inside a UDF or trigger does not contribute to the CONCURRENTDBCOORDACTIVITIES threshold.
- A cursor activity nested within a load activity does not contribute to the CONCURRENTDBCOORDACTIVITIES threshold.
Activities and the load utility

Running the load utility will generate several activities, one of which is a load activity and several others that are of type READ, WRITE, or OTHER. In the case of a load from cursor, an additional activity for the cursor the load activity is loading from is created. This cursor activity is a nested activity of the load activity.

Activities event monitoring

When monitoring activities with an activities event monitor in a multimember database environment, you must be aware of how activity event information is captured. If you are capturing activity events on all members, you might see more than one event for a given activity on members other than the coordinator. Multiple records per activity might be recorded because the activity might come and go on the remote members, depending on the sequencing of events in a section (the executable form of an SQL statement). As a result, multiple records might be captured for the activity on a non-coordinating member. To understand the processing performed by the activity on the remote member, you must consider all the records for the activity. For example, you might want to aggregate metrics from all records for the activity on the remote member.

By contrast, the activity would be recorded only once in the event data collected on the coordinating member.

Workload management DDL statements

Workload management DDL statements consist of the CREATE, ALTER, and DROP statements that you use to work with service classes, workloads, work class sets, work action sets, thresholds, and histograms.

The DB2 workload management DDL statements are as follows:

- CREATE SERVICE CLASS, ALTER SERVICE CLASS, and DROP SERVICE CLASS
- CREATE WORKLOAD, ALTER WORKLOAD, and DROP WORKLOAD
- GRANT USAGE ON WORKLOAD and REVOKE USAGE ON WORKLOAD
- CREATE THRESHOLD, ALTER THRESHOLD, and DROP THRESHOLD
- CREATE WORK CLASS SET, ALTER WORK CLASS SET, and DROP WORK CLASS SET
- CREATE WORK ACTION SET, ALTER WORK ACTION SET, and DROP WORK ACTION SET
- CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, and DROP HISTOGRAM TEMPLATE

Workload management DDL statements differ from other DB2 DDL statements:

- Only one uncommitted workload management DDL statement is permitted at a time across all database members. If an uncommitted workload management DDL statement exists, subsequent workload management DDL statements wait until the uncommitted workload management DDL statement is either committed or rolled back. Workload management DDL statements are processed in the order in which they are issued.
- Every workload management DDL statement must be followed by a COMMIT or ROLLBACK statement.
A workload management DDL statement cannot be issued in an XA transaction. After a connection issues a workload management DDL statement, the same connection must issue a COMMIT or ROLLBACK statement immediately after the workload management DDL statement. With XA transactions, it is possible for multiple connections to join a transaction, and any of the connections can commit or roll back the transaction. In this situation, it is impossible to ensure that the workload management environment would be correctly implemented.

DB2 for z/OS® does not recognize DB2 Database for Linux, UNIX, and Windows workload management DDL statements.

**Work identification by origin with workloads**

Workloads identify incoming work based on its source so that it can later be monitored or managed by DB2 workload management independently of all other work. The source is determined using the attributes of the database connection under which the work is submitted.

The attributes of a connection are first evaluated when the connection is established and the connection is matched to a workload definition. This relationship between a connection and a specific workload definition is called a **workload occurrence**. If any of the attributes change during the life of that connection, the workload assignment is reevaluated at the start of the next unit of work following the change. If a new workload definition is found to be a better match for the connection at this time, the old workload occurrence (for the previously assigned workload) ends and a new occurrence starts for the newly assigned workload definition. Although each connection is assigned to one and only one workload at a time, there can be multiple connections (workload occurrences) assigned to the same workload definition simultaneously. For more information, see “Workload assignment” on page 23.

For example, to assign all connections created by the application Accounts to a workload REPORTING, which maps the activities under those connections to run in the **Marketing** service class, issue a CREATE WORKLOAD statement such as the following:

```
CREATE WORKLOAD REPORTING APPLNAME('Accounts') SERVICE CLASS Marketing
```

This creates the following workload:

![User requests -> Workload REPORTING -> Marketing]

**Figure 2. The REPORTING workload**

Then grant the USAGE privilege on the REPORTING workload to PUBLIC:

```
GRANT USAGE ON WORKLOAD REPORTING TO PUBLIC
```

To assign all activities created by the application Accounts under the connections that belong to the session user group **Deptmgr** to the SUMMARY workload, which maps the activities to the **HumanResources** service class, issue a statement such as the following:

```
CREATE WORKLOAD SUMMARY SESSION_USER_GROUP('Deptmgr') APPLNAME('Accounts') SERVICE CLASS HumanResources
```
This creates the following workload:

```
GRANT USAGE ON WORKLOAD SUMMARY TO PUBLIC
```

You can view your workload definitions by querying the SYSCAT.WORKLOADS view, and you can view the connection attributes that you specified for each workload by querying the SYSCAT.WORKLOADCONNATTR view. You can view who is authorized to use a workload by querying the SYSCAT.WORKLOADAUTH view. To see what workload occurrences exist on the system at any point in time, use the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES or MON_GET_WORKLOAD table function.

SYSDEFAULTUSERWORKLOAD is the default workload. Any connection that is not assigned to a custom-defined workload during workload evaluation is assigned to this default workload, which ensures that all database connections are associated with a workload. Work assigned to the default workload SYSDEFAULTUSERWORKLOAD is executed in the SYSDEFAULTUSERCLASS service class by default.

**Supported database connection attributes**

You must specify at least one database connection attribute in the workload definition, and each connection attribute can have one or more values. If you do not specify a value for a specific connection attribute in the workload definition, the data server does not examine that attribute during workload evaluation.

<table>
<thead>
<tr>
<th>Connection attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>The actual communication address used by the client to communicate with the database server. The only protocol supported is TCP/IP. The address must be an IPv4 address, an IPv6 address, or a secure domain name.</td>
</tr>
<tr>
<td>Application name</td>
<td>The name of the application running at the client, as known to the data server. The application name is equivalent to the value shown in the Application name field in the system monitor output. See the <code>appl_name</code> monitor element for more information.</td>
</tr>
<tr>
<td>System authorization ID</td>
<td>The authorization ID of the user who connected to the database, as set in the SYSTEM_USER special register. You can change the value of SYSTEM_USER by connecting as a user with a different authorization ID.</td>
</tr>
<tr>
<td>Connection attribute</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Session authorization ID</td>
<td>The authorization ID that is used for the current session of the application, as set in the SESSION_USER special register. You can change the value of SESSION_USER by using a trusted context or the SET SESSION AUTHORIZATION statement.</td>
</tr>
<tr>
<td>Group of session authorization ID</td>
<td>The groups to which the current session user belongs.</td>
</tr>
<tr>
<td>Role of session authorization ID</td>
<td>The roles granted to the current session user.</td>
</tr>
<tr>
<td>Client user ID</td>
<td>The client user ID from the client information as set in the CURRENT_CLIENT_USERID (or CLIENT_USERID) special register. You can change the value of the client user ID by using one of the defined mechanisms provided by specific DB2 clients, the sqleseti (set client information) API, or the WLM_SET_CLIENT_INFO procedure.</td>
</tr>
<tr>
<td>Client application name</td>
<td>The application name from the client information as set in the CURRENT_CLIENT_APPLNAME (or CLIENT_APPLNAME) special register. You can change the value of the client application name by using one of the defined mechanisms provided by specific DB2 clients, the sqleseti API, or the WLM_SET_CLIENT_INFO procedure.</td>
</tr>
<tr>
<td>Client workstation name</td>
<td>The workstation name from the client information as set in the CURRENT_CLIENT_WRKSTNNAME (or CLIENT_WRKSTNNAME) special register. You can change the value of the client workstation name by using one of the defined mechanisms provided by specific DB2 clients, the sqleseti API, or the WLM_SET_CLIENT_INFO procedure.</td>
</tr>
<tr>
<td>Client accounting string</td>
<td>The accounting string from the client information as set in the CURRENT_CLIENT_ACCTNG (or CLIENT_ACCTNG) special register. You can change the value of the client accounting string by using the sqleseti API or the WLM_SET_CLIENT_INFO procedure.</td>
</tr>
</tbody>
</table>

### Use of wild cards in connection attributes

Some connection attributes support the specification of an asterisk (*) as a wild card in the CREATE WORKLOAD and ALTER WORKLOAD statements. You can use wild cards in situations where a connection attribute can take on several similar values, which can be matched by a regular expression using wild cards, without defining connection attributes for each of the possible values.
The wild card asterisk (*) matches zero or more characters. If you need to match an asterisk, use a double asterisk (**) to specify the asterisk as a literal character.

For example: If you have several accounts receivable applications (accrec01, accrec02 ... accrec15) that you all want to belong to the same workload for equal treatment by DB2 workload manager, define the CURRENT CLIENT_APPLNAME('accrec*') connection attribute to match all of these applications when you create or alter your workload. Similarly, an acc*rec accounts receivable application (a name that includes an asterisk character) is matched by the CURRENT CLIENT_APPLNAME('acc*rec') connection attribute.

The following workload connection attributes support the use of wild cards:

- APPLNAME
- CURRENT CLIENT_ACCTNG
- CURRENT CLIENT_APPLNAME
- CURRENT CLIENT_USERID
- CURRENT CLIENT_WRKSTNNAME

Set client information to identify requests

By default, many application servers set up connections with the same information and pass the same client information, if any, for all client requests that they handle. Some products, such as WebSphere and Cognos®, provide a facility to push down unique information about each request through the client information fields. This information uniquely identifies the end-user request within DB2. Most other products provide a way for the application server to be customized so that unique client information can be sent to DB2 prior to the start of processing an end-user request.

Specifying unique client attributes from the application server enables the specialized treatment of requests within DB2 and the assignment of requests from different clients to different workloads (and to different service classes).

Connection attribute evaluation order

As you analyze the usage characteristics of your environment, you can use the CREATE WORKLOAD statement to create your own workloads and map them to specific service classes. When you create the workload, you define both the values that are used to evaluate the connection attributes during workload assignment and the order in which the workload is evaluated relative to other workloads. Because more than one workload can match incoming connection attributes, being able to change the evaluation order enables you to determine which matching workload is chosen. Whether or not the session user has the USAGE privilege on the workload also determines which matching workload is chosen. For more information, see “Workload assignment” on page 23.

The following figure shows multiple requests being evaluated against workloads in the order A, B, C, and D, then assigned to specific workloads and executed in the applicable service class. Requests that cannot be matched to an existing workload are matched to the SYSDEFAULTUSERWORKLOAD workload. For information about the types of activities that run in the default maintenance class and default system class, see “Default service superclasses and subclasses” on page 70.
Workload assignment

At the beginning of the first unit of work after a database connection is established, the data server assigns the connection to a workload by evaluating the connection attributes of each workload that is enabled.

The order in which the workloads are evaluated is determined by the EVALUATIONORDER column value of each workload in the SYSCAT.WORKLOADS table. If a workload with matching connection attributes is found, the data server checks whether the current session user has the USAGE privilege on the workload. If the user has the USAGE privilege on the matching
workload, the workload assignment is complete, and the connection is assigned to that workload. If the user does not have the USAGE privilege on the matching workload, the data server continues to evaluate workloads until it finds a matching workload on which the session user has the USAGE privilege. If no matching workload is found, the data server attempts to use the SYSDEFAULTUSERWORKLOAD workload. If the current session user does not have the USAGE privilege on that workload, SQL4707N is returned, and the unit of work is rejected. Otherwise, the connection is assigned to the SYSDEFAULTUSERWORKLOAD workload.

You can set the evaluation order by using the POSITION keyword of the CREATE WORKLOAD or ALTER WORKLOAD statement, as follows:

- By specifying the absolute position of the workload in the evaluation order, as shown in the following example:

```
CREATE WORKLOAD...POSITION AT 2
```

POSITION AT 2 means that the workload is to be positioned second in the evaluation order. A matching workload that is positioned higher in the evaluation order is evaluated first. That is, if the workloads at both position 2 and position 3 match, the workload at position 2 is evaluated before the workload at position 3.

If the position that you specify on the CREATE WORKLOAD or ALTER WORKLOAD statement is greater than the total number of existing workloads, the workload is positioned next to last in the evaluation order, before the SYSDEFAULTUSERWORKLOAD workload. The effect is the same as specifying POSITION LAST on the CREATE WORKLOAD or ALTER WORKLOAD statement.

- By using the POSITION BEFORE `workload-name` or POSITION AFTER `workload-name` keyword, where `workload-name` is an existing workload. This keyword specifies the position of a new or altered workload relative to another workload in the evaluation order, as shown in the following example:

```
ALTER WORKLOAD...POSITION BEFORE workload2
```

If you do not specify the POSITION keyword, by default, the new workload is positioned after the other defined workloads in the evaluation order but before the SYSDEFAULTUSERWORKLOAD workload, which is always considered last.

**Workload reassignment**

A connection can potentially change its assigned workload at each unit of work boundary if one of its connection attributes or the set of workload definitions changes. A unit of work boundary is the point when a connection disassociates with its current transaction. The following events cause a unit of work boundary: Commit, rollback, XA end (success), XA commit, and XA rollback.

The workload assignment is reevaluated at the beginning of a new unit of work if the data server detects that one of the following events occurred:

- A relevant connection attribute changed. See the table in “Work identification by origin with workloads” on page 19 for a list of connection attributes that you can specify in a workload definition. Workload reevaluation also occurs if the current session authorization ID changes because the database connection switches because of a trusted context. For more information, see Trusted contexts and trusted connections.

- You created or altered a workload.
You granted the USAGE privilege on a workload to a user, a group, or a role or revoked the USAGE privilege on a workload from a user, group, or role.

A connection cannot be reassigned to a different workload while an activity that spans a unit of work boundary is still active. An activity can be an operation that maintains resources across multiple UOWs, such as a load operation, a stored procedure or table function, or a WITH HOLD cursor. The current workload occurrence runs until all activities complete. The workload reassignment then occurs at the beginning of the next unit of work.

An attempted workload assignment or reassignment results in an SQL4707N error if either of the following cases exists:

- The data server attempts to assign the connection to a workload that is dispermitted access to the database. For more information, see “Preventing occurrences of a workload from accessing the database” on page 36.
- The data server attempts to assign the connection to the SYSDEFAULTUSERWORKLOAD workload, but the current session user does not have the USAGE privilege on this workload.

If you have ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority, you can assign your database connection to the SYSDEFAULTADMWORKLOAD workload, the default administrator workload. See “Taking corrective action using the default administration workload” on page 28 for more information.

**XA transactions and workload reassignment**

XA calls such as XA_END (success), XA commit, and XA rollback issue a DB2 COMMIT or ROLLBACK, which indicates the end of a unit of work. Because workload reevaluation can occur at the beginning of a unit of work, these XA calls can initiate workload reevaluation, although the reason for workload reevaluation is not directly related to the XA transaction itself.

**Default workloads**

The default user workload SYSDEFAULTUSERWORKLOAD provides a workload for your data server to which all connections are assigned initially. The default administration workload SYSDEFAULTADMWORKLOAD permits you to take corrective administrative action that cannot otherwise be performed. Both workloads are created at database creation time and you cannot drop them.

**The default user workload (SYSDEFAULTUSERWORKLOAD)**

Connections that are assigned to the default user workload are mapped to the default user service superclass SYSDEFAULTUSERCLASS, which provides the default execution environment. You can map connections to user-defined service classes by creating user defined workloads. In addition, you can alter SYSDEFAULTUSERWORKLOAD so that it maps connections to a different service class than SYSDEFAULTUSERCLASS.

You can view the SYSDEFAULTUSERWORKLOAD workload by querying the SYSCAT.WORKLOADS table.

The following table shows shows the columns returned for the SYSDEFAULTUSERWORKLOAD workload in the SYSCAT.WORKLOADS view, along with values and whether you can modify these values. See “Workload
For more information, see SYSCAT.WORKLOADS.

The default administration workload (SYSDEFAULTADMWORKLOAD)

This workload permits ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM users to query the database and perform administrative or monitoring tasks at any time, but is typically used in cases when:

- The workload to which the administrator is assigned is not permitted to access the database (that is, the DISALLOW DB ACCESS keyword of the CREATE WORKLOAD or ALTER WORKLOAD statement was specified for the workload).
- A threshold was violated, preventing the administrator from performing work on the database.
The SYSDEFAULTADMWORKLOAD workload differs from other workloads in the following ways:

- You cannot drop or disable it.
- You cannot specify DISALLOW DB ACCESS for it.
- None of the thresholds apply to occurrences of this workload and the activities in it.
- You can run this workload only in the SYSDEFAULTUSERCLASS service superclass. See “Default service superclasses and subclasses” on page 70 for more information.
- You can assign a connection to this workload by using the SET WORKLOAD command from the CLP interface, or by invoking the WLM_SET_CLIENT_INFO stored procedure (and specifying SYSDEFAULTADMWORKLOAD for the client_workload parameter). For more information, see “Taking corrective action using the default administration workload” on page 28.

You can view the SYSDEFAULTADMWORKLOAD workload by querying the SYSCAT.WORKLOADS table. The following table shows the columns returned for the SYSDEFAULTADMWORKLOAD workload in the SYSCAT.WORKLOADS catalog view, along with values and whether you can modify these values:

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable using the ALTER WORKLOAD statement if you have DBADM or WLMADM authority (and SQLADM for COLLECT clauses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKLOADID</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>WORKLOADNAME</td>
<td>SYSDEFAULTADMWORKLOAD</td>
<td>No</td>
</tr>
<tr>
<td>EVALUATIONORDER</td>
<td>Last one</td>
<td>No</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>Timestamp of database creation</td>
<td>No</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>Timestamp of the last update of the workload definition</td>
<td>No (but the data server modifies this column when you update the workload definition)</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Y</td>
<td>No</td>
</tr>
<tr>
<td>ALLOWACCESS</td>
<td>Y</td>
<td>No</td>
</tr>
<tr>
<td>SERVICECLASSNAME</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>No</td>
</tr>
<tr>
<td>PARENTSERVICECLASSNAME</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>No</td>
</tr>
<tr>
<td>COLLECTAGGACTDATA</td>
<td>N</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTACTDATA</td>
<td>N</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTACTPARTITION</td>
<td>C</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTDEADLOCK</td>
<td>W</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTLOCKTIMEOUT</td>
<td>W</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTLOCKWAIT</td>
<td>N</td>
<td>Yes</td>
</tr>
<tr>
<td>LOCKWAITVALUE</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTACTMETRICS</td>
<td>N</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTUOWDATA</td>
<td>N</td>
<td>Yes</td>
</tr>
<tr>
<td>EXTERNALNAME</td>
<td>NULL</td>
<td>No</td>
</tr>
<tr>
<td>REMARKS</td>
<td>BLANK</td>
<td>Yes</td>
</tr>
</tbody>
</table>
For more information, see SYSCAT.WORKLOADS.

**Taking corrective action using the default administration workload**

The default administration workload SYSDEFAULTADMWORKLOAD is a special DB2-supplied workload definition that is not subject to any DB2 thresholds. Use this workload to take corrective action that cannot otherwise be performed, such as altering prohibitive threshold definitions that prevent all activities from running in a workload.

**Before you begin**

Use the SET WORKLOAD command (or the WLM_SET_CLIENT_INFO procedure) to assign a connection to the default administration workload SYSDEFAULTADMWORKLOAD.

Although you require no special authority to use the SET WORKLOAD command, you require ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority to assign a connection to the default administration workload. Otherwise, SQL0552N is returned during workload assignment.

**About this task**

Because this workload is not affected by thresholds, it has limited workload management control and is not recommended for use in submitting regular day-to-day work.

**Procedure**

To assign a connection to the default administration workload, issue the SET WORKLOAD command as follows:

```
SET WORKLOAD TO SYSDEFAULTADMWORKLOAD
```

When the command takes effect depends on when you issue it:

- If you issue the SET WORKLOAD TO SYSDEFAULTADMWORKLOAD command before the connection to the database, after the connection is established, it is assigned to SYSDEFAULTADMWORKLOAD at the beginning of the first unit of work.
- If you issue the SET WORKLOAD TO SYSDEFAULTADMWORKLOAD command at the beginning of a unit of work, after a connection to the database is established, the connection is assigned to SYSDEFAULTADMWORKLOAD when the first request that is not an sqleseti (Set Client Information) request is submitted.
- If you issue the SET WORKLOAD TO SYSDEFAULTADMWORKLOAD command at the middle of a unit of work, after a connection is established, the connection is assigned to SYSDEFAULTADMWORKLOAD at the beginning of the next unit of work.

When a connection is assigned to SYSDEFAULTADMWORKLOAD, workload reassignment is performed at the beginning of the next unit of work if either of the following situations occurs:

- You revoke SYSADM or DBADM authority from the session user. In this situation, SQL0552N is returned.
- You issue a SET WORKLOAD TO AUTOMATIC command. This command indicates that the next unit of work should not be assigned to the
SYSDEFAULTADMWORKLOAD workload and that a normal workload evaluation is to be performed at the beginning of the next unit of work. For more information, see “Workload assignment” on page 23.

Example

The following example shows how you can use the SYSDEFAULTADMWORKLOAD workload to take corrective action when no other corrective action is possible.

If you create a severely prohibitive concurrency threshold so that no activities can execute, because the threshold is always being exceeded, the same threshold can prevent you from correcting the problem. To be able to alter the prohibitive threshold, you must first set the workload so that the work runs in the default administration workload. Because activities running in this workload are not subject to thresholds, you can correct the problem and set the workload (for your ID) back to the default behavior.

The threshold that is the cause of the problem is created accidentally with the following statement. Concurrency should have been set to 100 but was set to 0. This threshold effectively prevents any activity from executing:

```
CREATE THRESHOLD PROHIBITIVE FOR DATABASE ACTIVITIES
  ENFORCEMENT DATABASE WHEN CONCURRENTDBCOORDACTIVITIES > 0
  STOP EXECUTION
```

Note: This statement is intended only to show you how a severely prohibitive threshold might be created. You should not issue this statement.

If you attempt to execute even just a simple SELECT statement, an error is returned, because concurrency is set to 0:

```
SELECT * FROM SYSCAT.TABLES
```

SQL4712N The threshold "PROHIBITIVE" has been exceeded. Reason code = "6".
SQLSTATE=5U026

Before you can take corrective action, you must set the workload to the default administration workload:

```
SET WORKLOAD TO SYSDEFAULTADMWORKLOAD
```

This statement can be issued only by someone with ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority and causes any connection to be assigned to the SYSDEFAULTADMWORKLOAD workload, where activities are not subject to the prohibitive threshold.

The problem can now be corrected by altering the threshold so that activities can run:

```
ALTER THRESHOLD PROHIBITIVE WHEN CONCURRENTDBCOORDACTIVITIES > 100 STOP EXECUTION
```

Once corrected, change the workload back so that the connection will no longer be assigned to SYSDEFAULTADMWORKLOAD but to whatever workload it was assigned to before:

```
SET WORKLOAD TO AUTOMATIC
```

The same SELECT statement used before should now complete successfully:
Creating a workload

Use a CREATE WORKLOAD statement to add a workload to the catalogs.

Before you begin

To create a workload, you require WLMADM or DBADM authority.

See the following topics for more information about prerequisites:

- “Workload management DDL statements” on page 18
- Appendix A, “General naming rules,” on page 533

Procedure

To create a workload:

1. Specify one or more of the following properties for the workload using the CREATE WORKLOAD statement:
   - The name of the workload.
   - The connection attributes. The incoming connection must supply matching connection attributes to those that you specified for the workload for a match to occur. For more information, see “Work identification by origin with workloads” on page 19. When specifying the connection attributes, note that values are ORed and attributes are ANDed: for example, UserID (bob OR sue OR frank) AND Application (SAS).
   - A value that indicates whether occurrences of this workload are permitted to access the database. By default, occurrences of this workload are permitted to access the database.
   - A value that indicates whether the workload is enabled or disabled. By default, the workload is enabled.
   - The service class where work submitted by occurrences of this workload is to be executed. The SYSDEFAULTUSERCLASS service superclass is the default. If you specify a user-defined service superclass, the work submitted by the workload occurrences runs in the SYSDEFAULTSUBCLASS service subclass of the service superclass.

   Note: You cannot specify the SYSDEFAULTSUBCLASS service subclass under any service superclass, including the SYSDEFAULTUSERCLASS service superclass.

   If you have defined a workload to point to a service superclass but do not want work submitted by it to run in the default SYSDEFAULTSUBCLASS service subclass, you can either change the workload definition to point directly to a user-defined service subclass or you can use a work action set defined on the service superclass to map individual pieces of work to a different service subclass. For more information, see “Work actions and work action sets” on page 94.
   - The position of the workload relative to other workloads when cached in the memory. The position of the new workload determines the order in which it is evaluated during workload assignment. By default, the new workload is
positioned last, which means that it is evaluated last, immediately before the default user workload is considered. For more information, see “Workload assignment” on page 23.

- The monitoring activity metrics collection level for activities submitted by connections associated with this workload. The default activity metrics collection setting for a workload is NONE. Note that the effective activity collection setting for activities is the combination of both the workload activity metrics collection level and the `mon_act_metrics` database configuration parameter.

- The type of activity information to collect. By default, no information for activities associated with the workload is sent to an activities event monitor.

- The aggregate activity information to collect. The aggregate activity information used for the workload only changes after the CREATE WORKLOAD operation is committed.

- The lock timeout events information to collect. By default, data about a lock event is sent to the locking event monitor, if one is active, when the lock event occurs, but previous lock timeout events are not sent (WITHOUT HISTORY).

- The deadlock information to collect. By default, data about a deadlock event is sent to the locking event monitor, if one is active, when the deadlock event occurs, but previous deadlock events are not sent (WITHOUT HISTORY).

- The lock wait information to collect. By default, no lock wait information is collected if a lock is not acquired within the set wait time.

- The unit of work information for each transaction associated with this workload to send to the unit of work event monitor, if one is active, when a unit of work ends. By default, no unit of work information is sent.

- The histogram templates that the workload should use as templates for its histograms. The histogram templates specified are reflected in the `SYSCAT.HISTOGRAMTEMPLATEUSE` view. For more information about histograms and histogram templates, see “Histograms in workload management” on page 258.

2. Commit your changes. When you commit your changes the workload is added to the `SYSCAT.WORKLOADS` view. Committing the change causes a workload re-evaluation to take place at the beginning of the next unit of work of each application. Depending on which workload is chosen, the application might be reassigned to a different workload.

**What to do next**

After you create a workload, you might need to grant the USAGE privilege on it to one or more session users. (Session users with WLMADM or DBADM authority have an implicit privilege to use any workload.) Even if a connection provides an exact match to the connection attributes of the workload, if the session user does not have the USAGE privilege on the workload, the data server does not consider the workload when performing workload evaluation. For more information, see “Granting the USAGE privilege on a workload” on page 36.

**Altering a workload**

An ALTER WORKLOAD statement changes a workload in the catalogs.
Before you begin

To alter a workload, you require SQLADM, WLMADM, or DBADM authority. To specify any clause other than a COLLECT clause, the authorization id must include WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

Procedure

To alter a workload:

1. Specify one or more of the following properties for the workload using the ALTER WORKLOAD statement:
   - The connection attributes. You can add connection attributes to and drop connection attributes from the workload definition unless it is the SYSDEFAULTUSERWORKLOAD or SYSDEFAULTADMWORKLOAD workload. The incoming connection must supply matching connection attributes to those that you specified for the workload for a match to occur. For more information, see “Work identification by origin with workloads” on page 19. To see the connection attributes for a workload, query the SYSCAT.WORKLOADCONNATTR view.
   - A value that indicates whether an occurrence of this workload is permitted to access the database. By default, an occurrence of this workload is permitted to access the database. You cannot remove database access from the SYSDEFAULTADMWORKLOAD workload.
   - A value that indicates whether the workload is enabled or disabled. By default, the workload is enabled. You cannot disable the SYSDEFAULTUSERWORKLOAD or the SYSDEFAULTADMWORKLOAD workload.
   - The service class under which occurrences of this workload are to be executed. The SYSDEFAULTUSERCLASS service superclass is the default. If you specify a user-defined service superclass, you can specify a service subclass under the service superclass. You cannot specify the SYSDEFAULTSUBCLASS subclass under any service superclass, including the SYSDEFAULTUSERCLASS service superclass. In addition, you cannot specify the SYSDEFAULTSYSTEMCLASS or SYSDEFAULTMAINTENANCECLASS service superclass.
   - The position of the workload relative to other workloads, which determines the order in which the workload is evaluated during workload assignment. You cannot specify the position of the SYSDEFAULTUSERWORKLOAD or the SYSDEFAULTADMWORKLOAD workload. For more information, see “Workload assignment” on page 23.
   - The type of activity information to collect. By default, no information for activities associated with the workload is sent to an activities event monitor.
   - The monitoring activity metrics collection level for activities submitted by connections associated with this workload. Note that the effective activity collection setting for activities is the combination of both the workload activity metrics collection level and the mon_act_metrics database configuration parameter.
   - The aggregate activity information to collect. The aggregate activity information used for the workload only changes after the ALTER WORKLOAD operation is committed.
• The lock timeout event information to send to the locking event monitor, if one is active, when a lock event occurs.
• The deadlock information to send to the locking event monitor, if one is active, when a deadlock event occurs.
• The lock wait information to collect.
• The unit of work information for each transaction associated with this workload to send to the unit of work event monitor, if one is active, when a unit of work ends.
• The histogram templates that the workload should use as templates for its histograms. The histogram templates specified are reflected in the SYSCAT.HISTOGRAMTEMPLATEUSE view. For more information about histograms and histogram templates, see “Histograms in workload management” on page 258.

2. Commit your changes. When you commit your changes the workload is updated in the SYSCAT.WORKLOADS view. The committed change causes a workload re-evaluation to take place at the beginning of the next unit of work of each application. Depending on which workload is chosen, the application might be reassigned to a different workload.

What to do next

You might need to grant the USAGE privilege on it to one or more session users. (Session users with DBADM authority have an implicit privilege to use any workload.) Even if a connection provides an exact match to the connection attributes of the workload, if the session user does not have the USAGE privilege on the workload, the data server does not associate the connection with the workload to create an occurrence of the workload. For more information, see “Granting the USAGE privilege on a workload” on page 36.

Enabling a workload

The DB2 data server checks the connection attributes specified for a workload against the connection attributes of the current session. The data server does not consider a disabled workload when it looks for a matching workload.

Before you begin

To alter a workload, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

About this task

By default, a workload is enabled when you create it. If you create a workload as disabled, you must enable it for the data server to consider the workload when it performs workload evaluation.

Procedure

To enable a workload:

1. Identify the workload that you want to enable. You can display the set of disabled workloads by querying the SYSCAT.WORKLOADS view, as shown in the following example:

   SELECT * FROM SYSCAT.WORKLOADS WHERE ENABLED='N'
2. Use the ALTER WORKLOAD statement to enable the disabled workload:

   ALTER WORKLOAD...ENABLE

   If the ALTER WORKLOAD statement is successful, the definition for the
   workload is written to the database catalog.

3. Commit your changes. When you commit your changes the workload is
   updated in the SYSCAT.WORKLOADS view.

Results

Enabling a workload takes effect at the beginning of the next unit of work. At that
point, a workload reevaluation occurs, and the data server considers the newly
enabled workload when it performs workload reevaluation.

Disabling a workload

Use this task to prevent specific workloads from being considered during
workload assignment. If you disable a workload, the data server does not consider
it when it looks for a matching workload. Instead, the data server assigns the unit
of work to the next matching workload. If no custom-defined workload matches,
the work is assigned to the default workload.

Before you begin

To create or alter a workload, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information
about prerequisites.

Procedure

To disable a workload:

1. Use the DISABLE option of the ALTER WORKLOAD statement to disable the
   workload:

   ALTER WORKLOAD...DISABLE

2. Commit your changes. When you commit your changes, the workload is
   updated in the SYSCAT.WORKLOADS view.

Results

Disabling a workload takes effect at the beginning of the next unit of work. At that
point, a workload reevaluation occurs, and the connection is assigned to the next
enabled workload that matches the connection attributes and for which there is
authorization.

Dropping a workload

Dropping a workload removes it from the database catalog.

Before you begin

To drop a workload, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information
about prerequisites.
**Procedure**

To drop a workload:

1. Disable the workload by specifying the `ALTER WORKLOAD` statement. See “Disabling a workload” on page 34 for more information. Disabling the workload prevents new occurrences of the workload from being able to run against the database.

2. Ensure that no occurrences of this workload are running by using the `WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES` table function. For more information, see `WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES` table function.

   The `WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES` table function returns the application handles corresponding to the active workload occurrences. You can use the `FORCE APPLICATION` command to terminate the applications using the application handles.

3. Drop the workload by specifying the `DROP WORKLOAD` statement. For example, to drop the `ACCTNG` workload, specify the following statement:

   ```sql
   DROP WORKLOAD ACCTNG
   ```

4. Commit your changes. When you commit your changes, the workload is removed from the `SYSCAT.WORKLOADS` view. In addition, authorization information for the workload is removed from the `SYSCAT.WORKLOADAUTH` view.

**Permitting occurrences of a workload to access the database**

If you have a workload that is not permitted to access the database but now want to permit occurrences of that workload to run, alter the workload so that it is permitted to access the database. By default, when a workload is created, it is permitted to access the database.

**Before you begin**

To alter a workload so that it can access a database, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

**About this task**

When you prevent a workload from accessing the database, the data server still examines that workload when performing workload assignment. However, all occurrences of that workload are rejected with an error.

**Procedure**

To permit a workload to access the database:

1. Use the `ALLOW DB ACCESS` option of the `ALTER WORKLOAD` statement to permit the workload to access the database. For example, to permit a workload called `WL1` to access the database, specify the following statement:

   ```sql
   ALTER WORKLOAD WL1 ALLOW DB ACCESS
   ```

2. Commit your changes. When you commit your changes workload is updated in the `SYSCAT.WORKLOADS` view.
Results

Altering a workload to permit its occurrences to access the database takes effect when the data server analyzes the next unit of work for that workload. For example, if you specified DISALLOW DB ACCESS for workload A and alter the workload by specifying ALLOW DB ACCESS, new occurrences of workload A are permitted to execute. Previously, any occurrence of workload A would have been rejected with an error.

Preventing occurrences of a workload from accessing the database

Use this task to control which workloads can access the database. Before a workload occurrence begins to run, the data server checks whether the workload is permitted to access the database. If you dispermit the workload occurrence from accessing the database, an error is returned indicating that the workload occurrence is rejected.

Before you begin

To prevent a workload from accessing the database, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

About this task

Preventing a workload occurrence differs from disabling a workload. When you disable a workload, the workload definition is not cached in memory and is therefore not considered for workload assignment.

Procedure

To prevent a workload from accessing a database:
1. Use the DISALLOW DB ACCESS option of the ALTER WORKLOAD statement, as shown in the following example:
   
   ```sql
   ALTER WORKLOAD workload-name DISALLOW DB ACCESS ... 
   ```

2. Commit your changes. When you commit your changes, the workload is updated in the SYSCAT.WORKLOADS view.

Results

Altering a workload to prevent its occurrences from accessing a database takes effect at the beginning of the next unit of work for workload occurrences that are already running. For example, if you specify ALLOW DB ACCESS for workload A and alter the workload by specifying DISALLOW DB ACCESS, occurrences of workload A that are already running receive an SQL error at the beginning of the next unit of work. New occurrences of workload A are rejected.

Granting the USAGE privilege on a workload

For a workload to be associated with a connection, the session user must have the USAGE privilege on that workload. Users with the ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority implicitly have the USAGE privilege on all workloads.
Before you begin

To use the GRANT USAGE ON WORKLOAD statement, you require ACCESSCTRL, SECADM, or WLMADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

About this task

When the data server finds a workload that matches the attributes of an incoming connection, the data server checks whether the session user has the USAGE privilege on that workload. If the session user does not have the USAGE privilege on that workload, the data server looks for the next matching workload. (In other words, the workloads for which the session user does not have the USAGE privilege are treated as if they do not exist.) Therefore, the workload USAGE privilege gives you the ability to further control which workload among the matching workloads a user, group, or role should be assigned to. For example, you can define more than one workload with the same connection attributes and grant the USAGE privilege on each of these workloads to only certain users, groups, or roles. For more information, see “Workload assignment” on page 23.

The client can set the client user ID, client application name, client workstation name, and client accounting string (which are some of the connection attributes that are used to assign a connection to a workload) without authorization. Therefore, the workload USAGE privilege also permits you to control which session user has the authority to use a workload.

You can view the USAGE privilege information by querying the SYSCAT.WORKLOADAUTH view.

If you create a database with the RESTRICTIVE option, the USAGE privilege on the SYSDEFAULTUSERWORKLOAD workload is not granted to PUBLIC at database creation time. You must explicitly grant the USAGE privilege on this workload to non-WLMADM and non-DBADM users. If the session user does not have the USAGE privilege on any of the workloads, including SYSDEFAULTUSERWORKLOAD, SQL4707N is returned when the data server attempts to associate a workload with the database connection.

Procedure

To grant the USAGE privilege on a workload:

1. Use the GRANT USAGE ON WORKLOAD statement. You can grant the USAGE privilege to specific users, groups, roles, or PUBLIC. For example, to grant the USAGE privilege on the ACCOUNTS workload to the CPA group, you would issue the following statement:

   ```sql
   GRANT USAGE ON WORKLOAD ACCOUNTS TO GROUP CPA
   ```

   You cannot grant the USAGE privilege on the SYSDEFAULTADMWORKLOAD workload. The SYSDEFAULTADMWORKLOAD workload can only be used by ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM users who issue the SET WORKLOAD TO SYSDEFAULTADMWORKLOAD command.

2. Commit your changes. When you commit your changes, the SYSCAT.WORKLOADAUTH view is updated. Until the GRANT statement is committed, the data server cannot consider the workload when performing workload assignment for the newly authorized users, groups, or roles.
**Revoking the USAGE privilege on a workload**

Use the `REVOKE USAGE ON WORKLOAD` statement to revoke the USAGE privilege on a workload.

**Before you begin**

To use the `REVOKE USAGE ON WORKLOAD` statement, you require `ACCESSCTRL`, `SECADM`, or `WLMADM` authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

**About this task**

You cannot explicitly revoke the USAGE privilege on the `SYSDEFAULTADMWORKLOAD` workload. Only `ACCESSCTRL`, `DATAACCESS`, `DBADM`, `SECADM`, or `WLMADM` users who issue the `SET WORKLOAD TO SYSDEFAULTADMWORKLOAD` command can use this workload. Therefore, `REVOKE USAGE ON WORKLOAD` statements do not work for `SYSDEFAULTADMWORKLOAD`.

**Procedure**

To revoke the USAGE privilege on a workload:

1. Use the `REVOKE USAGE ON WORKLOAD` statement. You can revoke the USAGE privilege from specific users, groups, roles, or PUBLIC. For example, to revoke the USAGE privilege on the `ACCOUNTS` workload from PUBLIC, you would specify the following statement:
   
   ```sql
   REVOKE USAGE ON WORKLOAD ACCOUNTS FROM PUBLIC
   ```

2. Commit your changes. When you commit your changes, the `SYSCAT.WORKLOADAUTH` view is updated. Until the `REVOKE` statement is committed, the data server considers the workload when performing workload assignment.

**Example: Workload assignment**

At the beginning of the first unit of work after a database connection is established, the data server assigns the connection to a workload by evaluating the connection attributes of each workload that is enabled.

Workload reevaluation occurs at the beginning of each unit of work if the value of a connection attribute or the workload definition itself changes during the unit of work.

The following figure shows a workload assignment. Users in the `Marketing` group who submit queries through `AppA` are assigned to the `APPAQUERIES` workload. They are not assigned to the `PAYROLL` workload, even though `PAYROLL` is positioned before `APPAQUERIES`, because the definition of workload `PAYROLL` specifies the `SESSION_USER GROUP` keyword as `Finance`. Users in the `Finance` group who submit queries through `AppA` are assigned to the `FINANCE` workload. They are not assigned to the `PAYROLL` workload, even though it is more specific and specifies both `AppA` and `Finance` in its definition, because the `FINANCE` workload is positioned before the `PAYROLL` workload. Users in the `Marketing` group who submit queries through `AppB` are assigned to the `SYSDEFAULTUSERWORKLOAD` workload, because none of the connection...
attributes specified in the FINANCE, PAYROLL, or APPAQUERIES workload definitions match the AppB application or Marketing group.

In the preceding figure, the CREATE WORKLOAD statements are as follows:

```
CREATE WORKLOAD PAYROLL APPLNAME ('AppA') SESSION_USER GROUP ('FINANCE') SERVICE CLASS SC1
CREATE WORKLOAD APPAQUERIES APPLNAME ('AppA') POSITION LAST SERVICE CLASS SC2
CREATE WORKLOAD FINANCE SESSION_USER GROUP ('FINANCE') SERVICE CLASS SC1 POSITION BEFORE PAYROLL
```

By default, many application servers set up connections with the same information and pass the same client information, if any, for all client requests that they handle. Some products, such as WebSphere and Cognos, provide a facility to push down unique information about each request through the client information fields. This information uniquely identifies the end-user request within DB2. Most other products provide a way for the application server to be customized so that unique client information can be sent to DB2 prior to the start of processing an end-user request.

Specifying unique client attributes from the application server enables the specialized treatment of requests within DB2 and the assignment of requests from different clients to different workloads (and to different service classes).
The following figure shows an example of a three-tier environment where queries are submitted by different user applications, (marketing.exe, auditing.exe, and reporting.exe), through an application server that establishes a connection to the database using the session user APPUSER. Three workloads are defined: one for queries submitted by marketing.exe, one for queries submitted by reporting.exe, and one for the rest of the queries. As shown in the figure, to assign queries submitted by marketing.exe to the MARKETING workload, the application server calls the sqleseti API to set the value of the CURRENT CLIENT_APPLNAME special register to marketing.exe. Similarly, to assign queries submitted by reporting.exe to the REPORTING workload, the server calls sqleseti to set the value of the CURRENT CLIENT_APPLNAME special register to reporting.exe.

Note that in the figure, when the server calls sqleseti to set the CURRENT CLIENT_USERID special register to Lidia (with nothing else changing; that is, the client application name is still set to reporting.exe), no workload reassignment occurs because there is no workload defined specifically with the CURRENT CLIENT_USERID set to Lidia.
The following statements are used to define the workloads specified in box 1 in the previous figure:

```
CREATE WORKLOAD MARKETING SESSION USER ('APPUSER')
CURRENT CLIENT_APPLNAME ('marketing.exe') SERVICE CLASS SC2
POSITION AT 1

CREATE WORKLOAD REPORTING SESSION USER ('APPUSER')
CURRENT CLIENT_APPLNAME ('reporting.exe') SERVICE CLASS SC4
```
POSITION AFTER MARKETING

CREATE WORKLOAD APPSERV SESSION_USER (‘APPUSER’)
SERVICE CLASS SCI

Example: Workload assignment when workload attributes have single values

The example in this topic shows how the data server performs workload assignment. In this example, only one value is specified for each workload connection attribute.

Assume that the following workloads exist in the catalog:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>ADDRESS</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_USERNAME</th>
<th>CURRENT_CLIENT_WKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REPORTS</td>
<td>AppA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INVENTORY REPORT</td>
<td></td>
<td>AppB</td>
<td>LYN</td>
<td>ACCOUNTING</td>
<td>TELMTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SALE REPORT</td>
<td>AppC</td>
<td></td>
<td>KATE</td>
<td>RATE</td>
<td>SALESREP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>AUDIT REPORT</td>
<td>AppB</td>
<td></td>
<td>ACCOUNTING</td>
<td>FINANALYST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>EXPENSE REPORT</td>
<td>AppA</td>
<td></td>
<td>TIM</td>
<td></td>
<td>EXPENSE APPROVER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>AUDIT RESULT</td>
<td>LYN</td>
<td></td>
<td>LYN</td>
<td></td>
<td>Audit Group</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assume that a database connection with the following attributes is established:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_USERNAME</th>
<th>CURRENT_CLIENT_WKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>166.53.111</td>
<td>AppA</td>
<td>TIM</td>
<td>TIM</td>
<td>FINANCE</td>
<td>FINANALYST, EXPENSE APPROVER</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>Business account</td>
</tr>
</tbody>
</table>

When the first unit of work is submitted, the data server checks each workload in the catalog, starting with the first workload in the list, and processes the workloads in ascending order until it finds a workload with matching attributes. When a matching workload is found, the unit of work runs under an occurrence of that workload. When determining which workload to assign the connection to, the data server compares the connection attributes in deterministic order.

The data server first checks the REPORTS workload for a match. The REPORTS workload is first in the list.

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>ADDRESS</th>
<th>APPLNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REPORTS</td>
<td>AppA</td>
<td></td>
</tr>
</tbody>
</table>

The data server checks the connection attributes in the following deterministic order:

1. APPLNAME. The value of APPLNAME, AppA, for the database connection matches the value of APPLNAME for the REPORTS workload.
2. SYSTEM_USER, which is not set in the workload definition. Any value (including a null value) is considered a match.
3. SESSION_USER, which is not set in the workload definition. Any value is considered a match.
4. SESSION_USER GROUP, which is not set in the workload definition. Any value is considered a match.
5. SESSION_USER ROLE, which is not set in the workload definition. Any value is considered a match.
6. CURRENT CLIENT_USERID, which is not set in the workload definition. Any value is considered a match.
7. CURRENT CLIENT_APPLNAME, which is not set in the workload definition. Any value is considered a match.
8. CURRENT CLIENT_WRKSTNNNAME, which is not set in the workload definition. Any value is considered a match.
9. CURRENT CLIENT_ACCTNG, which is not set in the workload definition. Any value is considered a match.

In this situation, because of the explicit and implicit matches between the connection attributes of the REPORTS workload and the information passed on the connection, the data server selects the REPORTS workload as a potential match. After selecting a workload, the data server then checks whether the session user has the USAGE privilege on the workload. Assuming that the session user TIM has the USAGE privilege on the REPORTS workload, that workload is used for the connection. If, however, TIM does not possess the USAGE privilege on the REPORTS workload, the data server continues by checking the INVENTORYREPORT workload for a match.

Assume that you want TIM to be assigned to the EXPENSEREPORT workload because that workload has additional connection attributes specified. In this situation, you would alter the evaluation order of the workloads to position EXPENSEREPORT before REPORTS in the workload list:

```
ALTER WORKLOAD EXPENSEREPORT POSITION AT 1
```

You could also use the following SQL statement to achieve the same result:

```
ALTER WORKLOAD EXPENSEREPORT BEFORE REPORTS
```

To ensure that the ALTER WORKLOAD statement takes effect, you must immediately issue a COMMIT statement after the ALTER WORKLOAD statement. The effect of the ALTER WORKLOAD statement on the catalog is as follows:

```
<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXPENSE REPORT</td>
<td>AppA</td>
<td>TIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>REPORTS</td>
<td>AppA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>INVENTORY REPORT</td>
<td>AppB</td>
<td>LYNNE</td>
<td>ACCOUNTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SALES REPORT</td>
<td>AppC</td>
<td>KATE</td>
<td>KATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>AUDIT REPORT</td>
<td>AppB</td>
<td>LYNNE</td>
<td>ACCOUNTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>AUDIT RESULT</td>
<td>AppB</td>
<td>LYNNE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

If TIM does not already have the USAGE privilege on the EXPENSEREPORT workload, you must issue the following statements (the COMMIT statement ensures that the GRANT statement takes effect):

```
GRANT USAGE ON WORKLOAD EXPENSEREPORT TO USER TIM
COMMIT
```

At the beginning of the next unit of work, workload reassignment occurs, and the data server assigns the connection from TIM to the EXPENSEREPORT workload.
In addition, new units of work submitted by other connections that have the same attributes are also associated with the EXPENSEREPORT workload.

**Example: Workload assignment for a unit of work when multiple workloads exist**

The example in this topic shows how the data server performs workload evaluation to assign the connection to an existing workload.

Assume that the following workloads are defined in the catalog:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXPENSE REPORT</td>
<td>AppB</td>
<td>TIM</td>
<td></td>
<td></td>
<td>EXPENSE APPROVER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>REPORTS</td>
<td>AppB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>INVENTORYREPORT</td>
<td>AppA</td>
<td>LYNN</td>
<td></td>
<td>ACCOUNTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SALES REPORT</td>
<td>AppC</td>
<td>KATE</td>
<td></td>
<td></td>
<td>SALESREP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>AUDIT REPORT</td>
<td>AppA</td>
<td></td>
<td></td>
<td>ACCOUNTING</td>
<td>FINANALIST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>AUDIT RESULT</td>
<td>LYNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Suppose that a database connection with the following attributes is established:

<table>
<thead>
<tr>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppA</td>
<td>LYNN</td>
<td>LYNN</td>
<td>ACCOUNTING</td>
<td>FINANALIST, SALESREP</td>
<td>LYNN</td>
<td>NULL</td>
<td>stress2</td>
<td>Audit group</td>
</tr>
</tbody>
</table>

When the first unit of work is submitted, the data server checks each workload in the catalog in ascending evaluation order and stops when it finds a workload whose connection attributes match those supplied by the connection. When it checks the workloads, the data server compares the connection attributes in deterministic order.

First, the data server checks the EXPENSE REPORT workload:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXPENSE REPORT</td>
<td>AppB</td>
<td>TIM</td>
<td></td>
<td></td>
<td>EXPENSE APPROVER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Because the APPLNAME attribute in the workload definition is AppB but the APPLNAME attribute passed by the connection is AppA, no match is possible. The data server proceeds to the REPORTS workload, which is second in the list:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>REPORTS</td>
<td>AppB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Again, the APPLNAME attribute in the workload definition is AppB, which does not match AppA. The data server proceeds to the third workload in the list, INVENTORYREPORT:
Table 13. INVENTORYREPORT workload in the catalog

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>INVENTORYREPORT</td>
<td>AppA</td>
<td>LYNN</td>
<td>ACCOUNTING</td>
<td>TELEMKTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The data server checks for a match between the submitted connection attributes and the INVENTORYREPORT workload. The attributes are checked in the following order:

1. APPLNAME. Both the workload definition and the connection have a value of AppA, so a match occurs.
2. SYSTEM_USER. Both the workload definition and the connection have a value of LYNN, so a match occurs.
3. SESSION_USER. The connection passed a value of LYNN. Because the SESSION_USER attribute is not set for the workload, any value, including a null value, that is passed by the connection matches.
4. SESSION_USER GROUP. Both the workload definition and the connection have a value of ACCOUNTING, so a match occurs.
5. SESSION_USER ROLE. The workload definition specifies the value TELEMKTR, but the connection supplied the values of FINANALYST and SALESREP. No match occurs for this attribute.

The data server stops trying to match the INVENTORYREPORT workload and the connection attributes and proceeds to the fourth workload in the list, SALESREPORT:

Table 14. SALESREPORT workload in the catalog

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>SALESREPORT</td>
<td>AppC</td>
<td>KATE</td>
<td>KATE</td>
<td>SALESREP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Because the APPLNAME of the SALESREPORT workload definition is AppC, no match occurs with the connection (which passed a value of AppA for APPLNAME). The data server then proceeds to the fifth workload in the list, AUDITREPORT:

Table 15. AUDITREPORT workload in the catalog

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>AUDITREPORT</td>
<td>AppA</td>
<td>ACCOUNTING</td>
<td>FINANALYST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The data server compares the attributes of the AUDITREPORT workload and the connection in the deterministic order:

1. APPLNAME. Both the workload definition and the connection have a value of AppA, so a match occurs.
2. SYSTEM_USER. The connection passed a value of LYNN. Because the SYSTEM_USER attribute is not set for the workload, any value passed by the connection matches.
3. SESSION_USER. The connection passed a value of LYNN. Because the SESSION_USER attribute is not set for the workload, any value passed by the connection matches.
4. SESSION_USER GROUP. Both the workload and the connection have a value of ACCOUNTING for this attribute, so a match occurs.
5. SESSION_USER ROLE. Both the workload and the connection have a value of FINANALYST for this attribute, so a match occurs.
6. CURRENT CLIENT_USERID. Because the CURRENT CLIENT_USERID attribute is not set for the workload, any value passed by the connection matches.

7. CURRENT CLIENT_APPLNAME. Because the CURRENT CLIENT_APPLNAME attribute is not set for the workload, any value passed by the connection matches.

8. CURRENT CLIENT_WRKSTNNAME. Because the CURRENT CLIENT_WRKSTNNAME attribute is not set for the workload, any value passed by the connection matches.

9. CURRENT CLIENT_ACCTNG. Because the CURRENT CLIENT_ACCTNG attribute is not set for the workload, any value passed by the connection matches.

After processing all the connection attributes and finding a matching workload, the data server checks whether the session user has the USAGE privilege on the workload. Assume that LYNN does not have the USAGE privilege on the AUDITREPORT workload. In this situation, although all of the connection attributes match, this workload is not associated with the connection. The data server proceeds to the sixth workload in the evaluation list, AUDITRESULT:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>AUDITRESULT</td>
<td>LYNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Audit Group</td>
</tr>
</tbody>
</table>

The data server compares the attributes of the AUDITRESULT workload and the connection in the deterministic order:

1. APPLNAME. Because the APPLNAME attribute is not set for the workload, any value passed by the connection matches.

2. SYSTEM_USER. Because the SYSTEM_USER attribute is not set for the workload, any value passed by the connection matches.

3. SESSION_USER. Both the workload and the connection have a value of LYNN for this attribute, so a match occurs.

4. SESSION_USER GROUP. Because the SESSION_USER GROUP attribute is not set for the workload, any value passed by the connection matches.

5. SESSION_USER ROLE. Because the SESSION_USER ROLE attribute is not set for the workload, any value passed by the connection matches.

6. CURRENT CLIENT_USERID. Both the workload and the connection have a value of LYNN for this attribute, so a match occurs.

7. CURRENT CLIENT_APPLNAME. Because the CURRENT CLIENT_APPLNAME attribute is not set for the workload, any value passed by the connection matches.

8. CURRENT CLIENT_WRKSTNNAME. Because the CURRENT CLIENT_WRKSTNNAME attribute is not set for the workload, any value passed by the connection matches.

9. CURRENT CLIENT_ACCTNG. Both the workload and the connection have a value of Audit Group for this attribute, so a match occurs.

After processing all of the connection attributes and finding a matching workload, the data server checks whether the session user has the USAGE privilege on the workload. In this situation, assume that the session user LYNN has the USAGE
privilege on the AUDITRESULT workload. Because all of the connection attributes match and the session user has the USAGE privilege, the connection is assigned to the AUDITRESULT workload.

Example: Workload assignment when workload attributes have multiple values

The example in this topic shows how the data server performs workload assignment. In this example, some of the workload definitions permit more than one value for a connection attribute.

Assume that the following workloads are defined in the catalog:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ITEMINQ</td>
<td>LINDA</td>
<td>KYLE, GEORGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DAILY TRANS REPORT</td>
<td>AppC</td>
<td>KYLE, CAROL</td>
<td>SALES, ACCOUNTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SALES SUMMARY</td>
<td>AppA, AppB</td>
<td></td>
<td>ACCOUNTANT, FINANALYST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assume that a database connection with the following attributes is established:

<table>
<thead>
<tr>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppC</td>
<td>LINDA</td>
<td>KYLE</td>
<td>SALES</td>
<td>ACCOUNTANT</td>
<td>NULL</td>
<td>NULL</td>
<td>Business Account</td>
<td></td>
</tr>
</tbody>
</table>

When the first unit of work is submitted, the data server checks each workload in the catalog in ascending evaluation order and stops when it finds a workload whose connection attributes match those supplied by the connection. When it checks the workloads, the data server compares the connection attributes in deterministic order.

First, the data server checks the ITEMINQ workload:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ITEMINQ</td>
<td>KYLE, GEORGE</td>
<td></td>
<td>RETAIL, SALES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The data server checks for a match between the submitted connection attributes and the ITEMINQ workload. The attributes are checked in the following order:

1. APPLNAME. Because the APPLNAME attribute is not set for the workload, any value, including a null value, that is passed by the connection matches.
2. SYSTEM_USER. The connection passed a value of LINDA. However, the ITEMINQ workload values are KYLE and GEORGE. No match occurs for this attribute.

The data server stops trying to match the ITEMINQ workload and the connection and proceeds to the second workload in the list, DAILYTRANSREPORT:

<table>
<thead>
<tr>
<th>Evaluation order</th>
<th>Workload name</th>
<th>APPLNAME</th>
<th>SYSTEM_USER</th>
<th>SESSION_USER</th>
<th>SESSION_USER_GROUP</th>
<th>SESSION_USER_ROLE</th>
<th>CURRENT_CLIENT_USERID</th>
<th>CURRENT_CLIENT_APPLNAME</th>
<th>CURRENT_CLIENT_WRKSTNNAME</th>
<th>CURRENT_CLIENT_ACCTNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>DAILYTRANSREPORT</td>
<td>AppC</td>
<td>KYLE, CAROL</td>
<td>SALES, ACCOUNTING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The data server compares the attributes of the DAILYTRANSREPORT workload and the connection in deterministic order:

1. APPLNAME. Both the workload definition and the connection have a value of AppC, so a match occurs.
2. SYSTEM_USER. Because the SYSTEM_USER attribute is not set for the workload, any value, including a null value, that is passed by the connection matches.
3. SESSION_USER. The SESSION_USER value passed on the connection is KYLE, which is a match with one of the workload SESSION_USER values. If the connection had passed CAROL, this would also be a match because both KYLE and CAROL are specified as part of the DAILYTRANSREPORT workload definition.
4. SESSION_USER GROUP. The SESSION_USER GROUP value passed on the connection is SALES, which matches the SALES value specified for the workload SESSION_USER GROUP attribute. If the connection had passed ACCOUNTING, this would also be a match because both SALES and ACCOUNTING are specified in the workload definition.
5. SESSION_USER ROLE. Because the SESSION_USER ROLE attribute is not set for the workload, any value passed by the connection matches.
6. CURRENT CLIENT_USERID. Because the CURRENT CLIENT_USERID attribute is not set for the workload, any value passed by the connection matches.
7. CURRENT CLIENT_APPLNAME. Because the CURRENT CLIENT_APPLNAME attribute is not set for the workload, any value passed by the connection matches.
8. CURRENT CLIENT_WRKSTNNAME. Because the CURRENT CLIENT_WRKSTNNAME attribute is not set for the workload, any value passed by the connection matches.
9. CURRENT CLIENT_ACCTNG. Because the CURRENT CLIENT_WRKSTNNAME attribute is not set for the workload, any value passed by the connection matches.

After processing all of the connection attributes and finding a matching workload for the connection, the data server checks whether the session user has the USAGE privilege on the workload. In this situation, assume that the session user KYLE has the USAGE privilege on the DAILYTRANSREPORT workload. Because all connection attributes match and the session user has the USAGE privilege, the connection is assigned to the DAILYTRANSREPORT workload.

Work identification by type, cost, or data accessed with work classes

In addition to using connection attributes that focus on the origin of activities with workloads, you can identify activities based on the type, cost or data that may be accessed through the creation of a work class set containing a work class.

A work class is a method of categorizing individual database activities based on attributes of the activities. If a work class has a work action defined for it, the work action will be applied to the work class and determines how the activities in the work class are managed. For more information, see “Apply controls to types of activities with work action sets” on page 91.

The following table shows the type keywords available for work classes and the SQL statements that correspond to the different keywords. Except for the load utility, all the statements in the following table are intercepted immediately before
execution in the processing of an EXECUTE, EXECUTE IMMEDIATE, or OPEN request. The load utility, when issued from a client, might issue requests before starting the actual load operation on the data server.

### Table 21. Work types

<table>
<thead>
<tr>
<th>Work type keyword</th>
<th>Applicable SQL statements</th>
</tr>
</thead>
</table>
| **READ, including SET statements with embedded READ SQL** | • All SELECT statements (select into, values into, full select)  
  **Exception:** SELECT statements containing a DELETE, INSERT, or UPDATE are not included.  
  • All XQuery statements |
| **WRITE, including SET statements with embedded WRITE SQL** | • All UPDATE statements (searched, positioned)  
  • All DELETE statements (searched, positioned)  
  • All INSERT statements (values, subselect)  
  • All MERGE statements  
  • All SELECT statements containing a DELETE, INSERT, or UPDATE statement |
| **CALL** | CALL statement  
  The CALL statement is only classified under the CALL and ALL work class types.  
  **Note:** Both anonymous blocks and autonomous routines are classified as CALL statements. |
| **DML, including SET statements with embedded READ or WRITE SQL** | All statements that are classified under the READ and WRITE work class types. |
| **DDL** | • All ALTER statements  
  • All CREATE statements  
  • COMMENT statement  
  • DECLARE GLOBAL TEMPORARY TABLE statement  
  • DROP statement  
  • FLUSH PACKAGE CACHE statement  
  • All GRANT statements  
  • REFRESH TABLE  
  • All RENAME statements  
  • All REVOKE statements  
  • SET INTEGRITY statement |
| **LOAD** | Load utility  
  The load utility is only classified under the LOAD and ALL work class types. |
Table 21. Work types (continued)

<table>
<thead>
<tr>
<th>Work type keyword</th>
<th>Applicable SQL statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Work types that are represented by all of the preceding keywords. <strong>Note:</strong> If the action is a threshold, the database activity that the threshold is applied to depends on the type of threshold. For example, if the threshold type is ESTIMATEDSQLCOST, only DML activity with an estimated cost (in timerons) is affected by the threshold. For more information, see “Example: Working with a work class defined with the ALL keyword” on page 63.</td>
</tr>
</tbody>
</table>

The following figure shows a hierarchical view of the work type keywords:

![Figure 7. Work type keywords](image)

SQL statements that do not fall under any of the available keywords are not classified, and behave as though no work class and work class set exists. For example, if the statement is SET SCHEMA and the only work class in the work class set has a work type of DML, that statement is not classified and no work action can be applied to it. So, if the action is MAP, the SET SCHEMA activity runs in the default service subclass (SYSDEFAULTSUBCLASS). If the action is a threshold, no threshold is applied to the activity.

**Additional identification**

Work classes also permit you to use predictive elements in the identification for DML work (or READ and WRITE statements). Predictive elements are useful because they provide information about database activities that can be used to take action before these activities start consuming resources on the data server. The following table provides information about predictive elements supported by work classes:

---

50 DB2 Workload Management Guide and Reference
Table 22. Characteristics for predictive identification

<table>
<thead>
<tr>
<th>Predictive element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated cost</td>
<td>Uses the estimated cost available from the DB2 compiler to include DML within a given timeron range (for example, create a work class for all large queries with an estimated cost over 1 000 000 timerons)</td>
</tr>
<tr>
<td>Estimated cardinality</td>
<td>Uses the estimated rows returned (cardinality) from the DB2 compiler to include DML within a given range of rows returned (for example, create a work class for large queries that are estimated to return more than 500 000 rows)</td>
</tr>
<tr>
<td>Estimated data accessed</td>
<td>Uses the estimated data accessed from the DB2 compiler to include DML that accesses data assigned a specific data tag (for example, create a work class for queries that are estimated to access data in table spaces tagged with a data tag value of 3)</td>
</tr>
</tbody>
</table>

Note: The set of table spaces that the DB2 compiler predicts will be accessed is based on the optimized SQL statement, which might differ from the user-specified SQL statement. In some cases, for example, when inserting into range partitioned tables, the number of table spaces that are predicted to be accessed is more than the expected table spaces.

You can also identify activities by using the schema name of the procedure that a CALL statement calls.

Based on workload attributes and work class types, you can identify work and prepare it for the next stage, the management of the work.

For more information on working with work classes and work class sets, refer the following topics:

**Work classes and work class sets**

A work class is a method of categorizing individual database activities based on attributes of the activities. Work classes are grouped into work class sets, which can be shared by different work action sets.

Examples of database activity attributes which can determine which work class an activity is associated with includes: activity type (DDL, DML, LOAD), the estimated cost (where available), the estimated cardinality (where available), the estimated data tag, and the schema (where available).

**Work classes**

A work class has the following attributes:

- The work class name, which must be unique in the work class set.
- The database activity attributes, which consist of the following information:
  - The type of database activity that falls into this work class. Using predefined keywords (for example, CALL, READ, WRITE, DML, DDL, LOAD, or ALL), you can classify database requests into different categories. Different types of database activities can be associated with a work class depending on its work type. For example, the WRITE keyword includes updates, deletes, inserts,
merges, and selects that contain a delete, insert, or update. For more
information, see “Work identification by type, cost, or data accessed with
work classes” on page 48.

– The range information that further categorizes DML or XQuery types of
database activity:
  - The type of range to specify (either timeron cost or cardinality). Specifying
    a range of values is optional. For example, when you specify a range for a
    work class, you can specify that all queries with an estimated cost of less
    than 100 timerons be processed differently than other queries.
  - The bottom of the range.
  - The top of the range.
– The schema of the routine to be called. Specifying the schema is optional.
  When defining a work class, you can use the schema attribute to further
  classify CALL statements according to the schema of the procedure being
  called. For example, if you specify SCHEMA1 for the schema of a work class
  and the work type is CALL, all CALL statements calling a SCHEMA1
  procedure are classified in that work class. If you specify the schema for a
  work class type other than CALL or ALL, the error SQL0628N is returned.
– The identification tag given to the data the activity might touch. For example,
  when you specify a data tag of 3 for a work class, activities that touch data in
  a table space or storage group with a data tag of 3 can be isolated and treated
differently.
  • The evaluation order of the work class (or position of the work class in the work
    class set). For more information, see “Evaluation order of work classes in a work
    class set” on page 58.
  • An automatically generated class identifier that uniquely identifies the work
    class.

You can create work classes in two ways:
• Create a new work class set to contain the new work class using the WORK
  CLASS keyword of the CREATE WORK CLASS SET statement.
• Add the new work class to an existing work class set using the ADD keyword of
  the ALTER WORK CLASS SET statement.

You can alter work classes by using the ALTER WORK CLASS keyword of the
ALTER WORK CLASS SET statement.

You can drop work classes from a work class set using the DROP WORK CLASS
keyword of the ALTER WORK CLASS SET statement, or by using the DROP
WORK CLASS SET statement to drop the work class set.

You can view your work classes by querying the SYSCAT.WORKCLASSES view.

**Work class sets**

You use work class sets to group one or more work classes. A work class set
consists of the following attributes:
• A unique descriptive name for the work class set
• Any comments that you want to supply for the work class set
• Zero or more work classes (although a work class can only exist in a work class
  set, a work class set does not have to contain any work classes)
• An automatically generated ID that uniquely identifies the work class set
You create a new work class set using the CREATE WORK CLASS SET statement. You can create an empty work class set and add work classes later, or you can create a work class set that contains one or more work classes.

You change an existing work class set in the following ways using the ALTER WORK CLASS SET statement:
- Add work classes to the work class set.
- Change work class attributes for work classes in the work class set.
- Drop work classes from the work class set.

You cannot change any work class set attributes.

Drop a work class set using the DROP WORK CLASS SET statement.

You can view your work class sets by querying the SYSCAT.WORKCLASSSETS catalog view.

The following figure shows an example of work classes in a work class set.

**Work class set: Large activities**

<table>
<thead>
<tr>
<th>Work class: Large reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT statements &gt; 1000000 (cardinality)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work class: Large writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE/INSERT/DELETE &gt; 20000 (timerons)</td>
</tr>
</tbody>
</table>

| Work class: Load |

*Figure 8. Example of work classes and a work class set*

For a work class set to be effective on the system, you must define a work action set and associate it with the work class set. By using a work action set, you can associate a work class set to a service superclass, a workload, or a database, to indicate what action should be applied to the database activities that fall within the classification. If you do not create a work action set for the work class set, the data server ignores the work class set.

**Creating a work class**

To create a work class, use the CREATE WORK CLASS SET statement or the ALTER WORK CLASS SET statement.

**Before you begin**

To create a work class, you require WLMADM or DBADM authority.

For additional prerequisites, see the following topics:
- “Workload management DDL statements” on page 18
• Naming rules

Procedure

To create a work class:

1. Create a work class at the same time you create a new work class set or add the new work class to an existing work class set:
   • To create a new work class that is added to a new work class set, use the WORK CLASS keyword of the CREATE WORK CLASS SET statement.
   • To create a new work class that is added to an existing work class set, use the ADD WORK CLASS keyword of the ALTER WORK CLASS SET statement.

Specify one or more of the following properties for the new work class:
• A name for the work class. This name must be unique in the work class set.
• Attributes for the work class. These attributes are used to associate an activity with the work class:
  – The type of work that the work class is to be used for. Use the WORK TYPE parameter to specify this characteristic.
    - READ, which represents non-updating SELECT activities, and all XQuery activities. When you specify the READ keyword, you can also specify an optional for-from-to-clause or data-tag-clause argument.
      • Use the optional for-from-to-clause argument to specify a range for either the cost of the statement in timerons, or its cardinality (that is, the number of rows returned). You must specify a numeric value for the first value. For the second value, you can specify either a numeric value, or the value UNBOUNDED to indicate that you do not want to impose an upper limit on either the cost or cardinality of the activity. You can also specify this argument for the WRITE keyword, the DML keyword, and the ALL keyword.
        For example, to associate SELECT activities that have a cost of 5000 timerons or more with this work class, you would specify:
        \[\text{WORK TYPE READ FOR TIMERONCOST FROM 5000 TO UNBOUNDED}\]
      • Use the optional data-tag-clause argument to specify a data tag identifying data that the activity is estimated to access. You can specify a value between 1 and 9. If the data-tag-clause is not specified no restriction is imposed on the type of data that a query accesses. You can also specify this argument for the WRITE keyword, the DML keyword, and the ALL keyword.
        For example, to associate SELECT activities with this work class, that are estimated to access data in table spaces that have been assigned a data tag value of 1, you would specify:
        \[\text{WORK TYPE READ DATA TAG LIST CONTAINS 1}\]
    - WRITE, which represents SQL activities that update data in the database. When you specify the WRITE keyword, you can also specify an optional for-from-to-clause or data-tag-clause argument.
      For example, to associate all data writing activities with this work class that, at compile time, are estimated to touch data tagged with a value of 5, you would specify:
      \[\text{WORK TYPE WRITE FOR CARDINALITY FROM 50 TO 100}\]
    - CALL, which represents CALL activities.
When you specify the CALL keyword, you can also specify the ROUTINES IN SCHEMA keyword to indicate that only CALL activities to routines in a specific schema should be associated with this work class. For example, if you only want to associate calls to routines in the ACCOUNTS schema to this work class, you would specify:

```
WORK TYPE CALL ROUTINES IN SCHEMA ACCOUNTS
```

- DML, which represents SQL activities covered by both the READ and WRITE keywords.

For example, to associate all DML activities that have an estimated cost in timerons from 500 to 1000 and, at compile time, are estimated to touch data tagged with a value of 8, you would specify:

```
WORK TYPE DML FOR TIMERONCOST FROM 500 TO 1000 DATA TAG LIST CONTAINS 8
```

- DDL, which represents the following activities:

  - ALTER
  - CREATE
  - COMMENT
  - DECLARE GLOBAL TEMPORARY TABLE
  - DROP
  - FLUSH PACKAGE CACHE
  - GRANT
  - REFRESH TABLE
  - RENAME
  - REVOKE
  - SET INTEGRITY

For example, to associate all DDL activities with this work class, you would specify:

```
WORK TYPE DDL
```

- LOAD, which represents a LOAD activity.

For example, to associate LOAD activities to this work class, you would specify:

```
WORK TYPE LOAD
```

- ALL, which represents all the work types indicated by all the preceding keywords.

When you specify ALL for a work class type, you can also specify the ROUTINES IN SCHEMA keyword to indicate that only CALL activities to routines in a specific schema should be associated with this work class. You can also specify the for-from-to-clause argument to indicate that all DML activities that have an estimated timeron cost or cardinality specified fall into this class. For example, to associate both DML activities that have a cardinality of 300 to 1500 rows and routines that are called from the NEWHIRES schema to this work class, you would specify the following statement. You can also specify the data-tag-clause argument to indicate all DML activities that access data in table spaces that are tagged with a data tag value. Because this work class has a type of ALL, it would also apply to other activities that do not have a schema or cardinality, such as LOAD activities and DDL activities.

```
WORK TYPE ALL FOR CARDINALITY FROM 300 TO 1500 ROUTINES IN SCHEMA NEWHIRES
```
Optional. The position of the work class in the work class set. The position of the work class in the work class set determines the order in which the work class is evaluated when classifying an activity to a work class. When work class assignment occurs, the data server first determines the work class set associated with the object (either a service superclass or the database), then selects the first matching work class in the work class set that has a work action associated with it. Use the POSITION keyword to specify one of the following:

- LAST. The work class is placed at the end of the list of work classes in the work class set. For example:
  WORK TYPE ... POSITION LAST
- BEFORE work-class-name. The work class is to be created in the work class set and positioned before the specified work class. For example:
  WORK TYPE ... POSITION BEFORE LARGEDDL
- AFTER work-class-name. The work class is to be created in the work class set and positioned after the specified work class. For example:
  WORK TYPE ... POSITION AFTER LARGEDDL
- AT integer. The work class is to be created in the work class set in the position specified by the integer value. For example:
  WORK TYPE ... POSITION AT 3

2. Commit your changes. When you commit your changes, the work class is added to the SYSCAT.WORKCLASSES view.

Assuming a work class

If you need to alter a work class, use the ALTER WORK CLASS SET statement.

Before you begin

To alter a work class, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for additional prerequisites.

Procedure

To alter a work class:

1. Use the ALTER keyword of the ALTER WORK CLASS SET statement to change one or more of the following properties. See “Creating a work class” on page 53 for an explanation of the supported values for these properties.

   - The FOR keyword. For example, you can change the value specified for the FOR keyword from CARDINALITY to TIMERONCOST.
   - The FROM from-value TO to-value argument. For example, you can change the argument from FROM 50 TO 100 to FROM 500 TO 1500.
   - The ROUTINES IN SCHEMA or the ROUTINES IN ALL keywords, for CALL activities. For example, if the work class currently does not specify a schema, you can add one. You can also specify the keyword ALL, so that the work class applies to all CALL statements, regardless of the schema of the routine. ALL is the default.
   - The DATA TAG LIST CONTAINS keyword. For example, you can change the value specified for this keyword from ANY to 8.
   - The POSITION keyword, followed by the keywords LAST, BEFORE, AFTER, or AT. If you specify POSITION BEFORE or POSITION AFTER, you must also specify the work class that you want to use to position your altered...
work class. If you specify POSITION AT, you must include the position number. For example, you can move a work class from the last position to any position by using the AT keyword, or from any position to the last position by using the LAST keyword.

2. Commit your changes. When you commit your changes, the work class is updated in the SYSCAT.WORKCLASSES view.

**Dropping a work class**

If you no longer require a work class, you can drop it from the work class set.

**Before you begin**

To drop a work class, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for additional prerequisites.

**Procedure**

To drop a work class:

1. Use the DROP keyword of the ALTER WORK CLASS SET statement. You cannot drop a work class if any work action in any work action set associated with this work class set has a dependency on the work class you want to drop. In this situation, you must first drop all dependent work actions before dropping the work class.
2. Commit your changes. When you commit your changes, the work class is removed from the SYSCAT.WORKCLASSES view.

**Creating a work class set**

To create a work class set, use the CREATE WORK CLASS SET statement.

**Before you begin**

To create a work class set, you require WLMADM or DBADM authority.

For additional prerequisites, see the following topics:
- “Workload management DDL statements” on page 18

**Procedure**

To create a work class set:

1. Specify the following properties for the work class set using the CREATE WORK CLASS SET statement:
   - A name for the work class set. The name you specify must be unique in the database.
   - Optional: One or more work classes for the work class set. For more information, see “Creating a work class” on page 53.
2. Commit your changes. When you commit your changes, the work class set is added to the SYSCAT.WORKCLASSSETS view.

**Altering a work class set**

You cannot change the work class set attributes after you create a work class set. However, you can add, alter, and drop work classes in the work class set using the ALTER WORK CLASS SET statement.
Before you begin

To alter a work class set, you require WLMADM or DBADM authority.

For additional prerequisites, see the following topics:
- “Workload management DDL statements” on page 18
- Naming rules

Procedure
1. If you want to add work class to the work class set, use the ADD keyword. For information about the keywords that you can specify when adding a work class, see “Creating a work class” on page 53.
2. If you want to alter a work class, use the ALTER keyword. For information about altering a work class, see “Altering a work class” on page 56.
3. If you want to drop a work class, use the DROP keyword. For information about dropping a work class from a work class set, see “Dropping a work class” on page 57. If you want to drop all the work classes from the work class set, you can drop the work class set itself. For more information, see “Dropping a work class set.”
4. Commit your changes. When you commit your changes, the SYSCAT.WORKCLASSES view is updated to show any added, altered, or dropped work class.

Dropping a work class set
Use the DROP WORK CLASS SET statement to drop a work class set.

Before you begin

To drop a work class set, you require WLMADM or DBADM authority.

About this task
You can drop a work class set only if no work action sets are associated with it. If you want to drop the work class set, you must first drop its dependent work action sets.

Procedure

To drop a work class set:
1. Use the DROP WORK CLASS SET statement.
2. Commit your changes. When you commit your changes the work class set is removed from the SYSCAT.WORKCLASSSETS view. In addition, all work classes that were part of the work class set are removed from the SYSCAT.WORKCLASSES view.

Evaluation order of work classes in a work class set
A work class set can have multiple work classes that match with a database activity. To select which work class from a work class set an activity should fall under, the data server goes through the work classes according to the evaluation order, stopping at the first work class that matches the activity.

If no matching work class exists, the database activity does not belong to any work class, and no work action is applied to that activity.
You can affect the evaluation order of work classes in a work class set when you create or alter a work class set. When you create or alter a work class set, you determine the position at which a work class is placed in the work class set using one of the following three methods:

- Specify the absolute position of the work class in the list.
  
  For example, POSITION AT 2. In this situation, the work class is placed in the second position in the work class set, and the work class that was at the second position is now the third, the third work class is now the fourth, and so on. If the position specified for the work class by the CREATE WORK CLASS SET or ALTER WORK CLASS SET statement is greater than the total number of work classes in the work class set, the work class is positioned last in the list.

- Use the POSITION BEFORE or POSITION AFTER keyword to specify the position of the work class relative to work classes already in the work class set.

- Omit the position when creating a work class.
  
  In this situation, the new work class is positioned at the end of the list. The position you specify for the work class in the work class set list is not necessarily the actual value of the EVALUATIONORDER column in the SYSCAT.WORKCLASSES view. The data server automatically assigns the order value to prevent gaps.

Work classes are processed in the order they are received, which can affect the evaluation order. For example, assume that you issue the following statement:

```
ALTER WORK CLASS SET WCS ALTER WORK CLASS C1 POSITION AT 1
ALTER WORK CLASS C2 POSITION AT 1
```

As a result, the C1 work class has a evaluation order of 2 and the C2 work class has an evaluation order of 1 because C2 was the last work class processed.

### Assignment of activities to work classes

If a work class set, through a work action set, is associated with a database, a workload, or a service superclass, then, just prior to execution in processing of an execute, execute immediate, or open request, or just before the execution of the load utility, the database activity is checked to determine if it matches any of the criteria specified in the work classes within the work class set.

The work classes are sorted within the work class set, by their evaluation order. Based on this evaluation order, the database activity is checked against each work class based on the attributes of the database activity (such as the activity type and cardinality) until there is a match or the list of work classes in the work class set has been exhausted.

Assume that the following work classes are in a work class set:

- Evaluation order: 1; work class name: MyLoad; work class type: LOAD
- Evaluation order: 2; work class name: SmallRead; work class type: READ; other attributes: estimated cost < 300 timerons
- Evaluation order: 3; work class name: AllDML; work class type: DML
- Evaluation order: 4; work class name: LargeRead; work class type: READ; other attributes: estimated cost > 301 timerons
- Evaluation order: 5; work class name: MyDDL; work class type: DDL

If a SELECT statement with an estimated cost of 200 timerons is received, it is assigned to the SmallRead work class. If a DDL activity (such as CREATE TABLE) arrives, it is assigned the MyDDL work class. If a SELECT statement with an
estimated cost of 500 timerons arrives, it is assigned to the AllDML work class because AllDML is positioned before the LargeRead work class. For more information, see “Example: Working with a work class defined with the ALL keyword” on page 63.

Work classifications supported by thresholds

Although any of the threshold types that can be used in work actions can be associated with any work class, not all types of database activities are supported for all of those threshold types.

For example, if you create a work class for DDL, then associate that work class with an ESTIMATEDSQLCOST threshold work action, that threshold will not apply to any of the requests that are classified under DDL because DDL statements do not have an estimated cost. If you create a work class for ALL, then associate that work class with an ESTIMATEDSQLCOST threshold work action, although all database activities belong to the ALL work class, the threshold will only apply to the database activities that have an estimated cost.

The following tables show which work class categories are supported by which threshold types:

<table>
<thead>
<tr>
<th>Work Classification</th>
<th>&quot;ACTIVITYTOTALTIME threshold&quot; on page 129</th>
<th>&quot;ESTIMATEDSQLCOST threshold&quot; on page 130</th>
<th>&quot;CONCURRENTDBCOORDACTIVITIES threshold&quot; on page 139</th>
<th>&quot;CPU TIME threshold&quot; on page 136</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ, including SET statements with embedded READ SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WRITE, including SET statements with embedded WRITE SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CALL</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DML, including SET statements with embedded READ or WRITE SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DDL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LOAD</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ALL</td>
<td>Yes</td>
<td>Some</td>
<td>Some</td>
<td>Some</td>
</tr>
</tbody>
</table>

Note:
1. Activities that run within user-defined functions (UDFs) and that contain these work classifications are not affected by the CONCURRENTDBCOORDACTIVITIES threshold.

Table 24. Work classification supported by thresholds (continued)

<table>
<thead>
<tr>
<th>Work Classification</th>
<th>&quot;SQLROWSREAD threshold&quot; on page 134</th>
<th>&quot;SQLROWSRETURNED threshold&quot; on page 136</th>
<th>&quot;SQLTEMPSPACE threshold&quot; on page 137</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ, including SET statements with embedded READ SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WRITE, including SET statements with embedded WRITE SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CALL</td>
<td>No</td>
<td>No (see note)</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 24. Work classification supported by thresholds (continued) (continued)

<table>
<thead>
<tr>
<th>DML, including SET statements with embedded READ or WRITE SQL</th>
<th>SQLROWSREAD threshold</th>
<th>SQLROWSRETURNED threshold</th>
<th>SQTENSPACE threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

| DDL                                                          | Yes                    | No                         | No                   |
| LOAD                                                         | No                     | No                         | No                   |
| ALL                                                          | Some                   | Some                      | Some                 |

Note:
- Although the statements in the procedure called may return rows, because the rows are not returned as a result of the CALL statement they are not controlled by the SQLROWSRETURNED threshold.

Example: Analyzing workloads by activity type

You can use DB2 workload management table functions to examine the workloads in your environment according to the types of activities being run.

In some situations, you might be interested in the behavior of a certain type of activities, such as LOAD activities. For example, you can observe how many LOAD activities are currently in the system as follows:

```sql
SELECT COUNT(*)
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(CAST(NULL AS BIGINT), -2))
AS ACTS
WHERE ACTIVITY_TYPE = 'LOAD'
```

You can obtain a count of how many activities of a specific type have been submitted since the last reset of the DB2 workload management statistics by using the WLM_GET_WORK_ACTION_SET_STATS table function, as shown in the following example. Assume that the READCLASS and LOADCLASS work classes exist for activities of type READ and activities of type LOAD. The * represents all activities that do not fall into the READCLASS or LOADCLASS work class.

```sql
SELECT SUBSTR(WORK_ACTION_SET_NAME,1,18) AS WORK_ACTION_SET_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       SUBSTR(WORK_CLASS_NAME,1,15) AS WORK_CLASS_NAME,
       LAST_RESET,
       SUBSTR(CHAR(ACT_TOTAL),1,14) AS TOTAL_ACTS
FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS('', -2)) AS WASSTATS
ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME, PART
```

WORK_ACTION_SET_NAME  PART  WORK_CLASS_NAME  LAST_RESET  TOTAL_ACTS
----------------------  ----  ---------------  -----------  -----------
AdminActionSet         0     ReadClass       2005-11-25-18.52.49.343000 8
AdminActionSet         1     ReadClass       2005-11-25-18.52.49.343000 0
AdminActionSet         0     LoadClass       2005-11-25-18.52.49.343000 2
AdminActionSet         1     LoadClass       2005-11-25-18.52.49.343000 0
AdminActionSet         0     *               2005-11-25-18.52.49.343000 0
AdminActionSet         1     *               2005-11-25-18.52.49.343000 0

You can view the average lifetime of LOAD activities by creating a work action set to map LOAD activities to a specific service subclass. For example, suppose you map LOAD activities to the service subclass LOADSERVICECLASS under the service superclass MYSUPERCLASS. Then, you can query the WLM_GET_SERVICE_SUBCLASS_STATS table function:

```sql
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       CAST(COORD_ACT_LIFETIME_AVG / 1000 AS DECIMAL(9,3)) AS AVG_LIFETIME
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('LOADSERVICECLASS', 'MYSUPERCLASS'))
```

Chapter 2. Work identification
Example: Using a work class set to manage specific types of activities

The following example shows how to use a work class set to manage DML activities.

Assume that you have a large number of applications running on your NONAME database each day and lately a few performance issues have been occurring. To deal with some of these issues, you decide that you need to be able to control the number of large queries (that is, any query that has an estimated cost of greater than 9999 timerons or an estimated cardinality of greater than 9999 rows) that can run simultaneously on the database.

To control the number of large queries that can run on the database, you would do the following:

1. Create a MYWORKCLASSSET work class set that contains two work classes:
   one for queries with a large estimated cost and one for queries with a large estimated cardinality. For example:
   
   CREATE WORK CLASS SET MYWORKCLASSSET
   (WORK CLASS LARGEESTIMATEDCOST WORK TYPE DML
   FOR TIMERONCOST FROM 10000 TO UNBOUNDED,
   WORK CLASS LARGECARDINALITY WORK TYPE DML
   FOR CARDINALITY FROM 10000 TO UNBOUNDED)

2. Create a DATABASEACTIONS work action set that contains two work actions that are to be applied to the work classes in the MYWORKCLASSSET work class set at the database level
   
   CREATE WORK ACTION SET DATABASEACTIONS FOR DATABASE
   USING WORK CLASS SET LARGEQUERIES
   (WORK ACTION ONECONCURRENTQUERY ON WORK CLASS LARGEESTIMATEDCOST
   WHEN CONCURRENTDBCOORDACTIVITIES > 1 AND QUEUEDACTIVITIES > 1 STOP EXECUTION,
   WORK ACTION TWOCONCURRENTQUERIES ON WORK CLASS LARGECARDINALITY
   WHEN CONCURRENTDBCOORDACTIVITIES > 2 AND QUEUEDACTIVITIES > 3 STOP EXECUTION)

In addition, several large administrative applications run daily against the database, and you want these applications to run in one resource pool. To accomplish this goal, you would create a service superclass called ADMINAPPS for these applications. For each application, you would create a workload to map it to the ADMINAPPS service superclass.

Because it is important that the queries (SELECT statements) run quickly, you decide to create a service subclass called SELECTS in the ADMINAPPS service superclass for these queries.

To map the SELECT statements to the SELECTS service subclass:

1. Create a SELECTDML work class set that contains a work class for all SELECT statements that do not update the database:
Create work class set SELECTDML (work class SELECTCLASS work type read)

2. Create an ADMINAPPSACTIONS work action set. This work action set contains a work action that is to be applied to the work class in work class set SELECTDML at the service superclass level.
CREATE WORK ACTION SET ADMINAPPSACTIONS FOR SERVICE CLASS ADMINAPPS USING WORK CLASS SET SELECTDML
(WORK ACTION MAPSELECTS ON WORK CLASS SELECTCLASS MAP ACTIVITY TO SELECTS)

Example: Working with a work class defined with the ALL keyword

This example shows how to work with a work class defined as ALL, which potentially covers all recognized activities in the database.

The DB2 workload management infrastructure recognizes the following specific subset from the domain of all possible database activities: load operations, the CALL statement, all DDL statements, and all DML statements. Full support for monitoring and controlling these recognized activities is provided by DB2 workload management.

When a work class with the type of ALL is used with a mapping work action, all recognized database activity is mapped to the service subclass specified in the work action. If a work class with the work type of ALL is used with a threshold work action, the threshold type determines which database activities the threshold applies to. Consider the following example.

Assume that you create a work class set called Example with the following work classes. The evaluation order of the work class is as follows:
1. SMALLDML, which is for all DML-type SQL that has an estimated cost of less than 1000 timerons.
2. LOADUTIL, which is for the load utility.
3. ALLACTIVITY, which is for all database activity

ALLACTIVITY is the last work class evaluated, and covers database activities that do not correspond to the first three work classes.

The DDL for creating this work class set is:
CREATE WORK CLASS SET EXAMPLE
(WORK CLASS SMALLDML WORK TYPE DML FOR TIMERONCOST FROM 0 TO 999,
WORK CLASS LOADUTIL WORK TYPE LOAD,
WORK CLASS ALLACTIVITY WORK TYPE ALL)

Assume that you have a service superclass called EXAMPLESERVICECLASS, and it has two service subclasses called SMALLACTIVITY and OTHERACTIVITY. You want to set up the system so that all small database activities run in the SMALLACTIVITY service subclass, and all other recognized database activities, except for the load utility, run in the OTHERACTIVITY service subclass. You do not want to remap the load utility to any other service subclass, but instead want it to run in the default service subclass.

To accomplish these goals, you would set up a work action set, SERVICECLASSACTIONS for the EXAMPLESERVICECLASS service superclass. The SERVICECLASSACTIONS work action set would contain the following work actions.
### Table 25. SERVICECLASSACTIONS work action set

<table>
<thead>
<tr>
<th>Work action</th>
<th>Work class applied to</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPDML</td>
<td>SMALLDML</td>
<td>Maps to the SMALLACTIVITY service subclass</td>
</tr>
<tr>
<td>COUNTLOAD</td>
<td>LOADUTIL</td>
<td>Counts the number of LOAD activities</td>
</tr>
<tr>
<td>MAPOTHER</td>
<td>ALLACTIVITY</td>
<td>Maps to the OTHERACTIVITY service subclass</td>
</tr>
</tbody>
</table>

The DDL to create this work action set is:

```
CREATE WORK ACTION SET SERVICECLASSACTIONS FOR SERVICE CLASS EXAMPLESERVICECLASS USING WORK CLASS SET EXAMPLE
(WORK ACTION MAPDML ON WORK CLASS SMALLDML MAP ACTIVITY TO SMALLACTIVITY,
 WORK ACTION COUNTLOAD ON WORK CLASS LOADUTIL COUNT ACTIVITY,
 WORK ACTION MAPOTHER ON WORK CLASS ALLACTIVITY MAP ACTIVITY TO OTHERACTIVITY)
```

Using this configuration, all small DML runs under the SMALLACTIVITY service subclass. The COUNTLOAD work action is applied to the LOADUTIL work class, which runs under the default service subclass. All other recognized database activities run under the OTHERACTIVITY service subclass.

**Note:** If the ALLACTIVITY work class were at the top of the evaluation order, all recognized activities would be mapped to the OTHERACTIVITY service subclass.

Now assume that you want to define a work action set for the database and apply thresholds that control what is permitted to run concurrently on the system. You could create a work action set called DATABASEACTIONS that contains the following work actions. The DML for creating this work action set is:

```
CREATE WORK ACTION SET DATABASEACTIONS FOR DATABASE USING WORK CLASS SET EXAMPLE
(WORK ACTION CONCURRENTSMALLDML ON WORK CLASS SMALLDML
 WHEN CONCURRENTDBCOORDACTIVITIES > 1000 AND QUEUEDACTIVITIES > 10000
 COLLECT ACTIVITY DATA STOP EXECUTION,
 WORK ACTION CONCURRENTLOAD ON WORK CLASS LOADUTIL
 WHEN CONCURRENTDBCOORDACTIVITIES > 2 AND QUEUEDACTIVITIES > 10
 COLLECT ACTIVITY DATA STOP EXECUTION,
 WORK ACTION CONCURRENTOTHER ON WORK CLASS ALLACTIVITY
 WHEN CONCURRENTDBCOORDACTIVITIES > 100 AND QUEUEDACTIVITIES > 100
 COLLECT ACTIVITY DATA STOP EXECUTION,
 WORK ACTION MAXCOSTALLOWED ON WORK CLASS ALLACTIVITY
 WHEN ESTIMATESQLCOST > 1000000 COLLECT ACTIVITY DATA STOP EXECUTION)
```

### Table 26. DATABASEACTIONS work action set

<table>
<thead>
<tr>
<th>Work action</th>
<th>Work class applied to</th>
<th>Threshold type and value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCURRENTSMALLDML</td>
<td>SMALLDML</td>
<td>Concurrency up to 1000 statements; queue up to 10 000 statements</td>
<td>• Stop execution&lt;br&gt;• Collect activity data</td>
</tr>
<tr>
<td>CONCURRENTLOAD</td>
<td>LOADUTIL</td>
<td>Concurrency up to 2 occurrences; queue up to 10 occurrences</td>
<td>• Stop execution&lt;br&gt;• Collect activity data</td>
</tr>
<tr>
<td>CONCURRENTOTHER</td>
<td>ALLACTIVITY</td>
<td>Concurrency up to 100 activities; queue up to 100 activities</td>
<td>• Stop execution&lt;br&gt;• Collect activity data</td>
</tr>
</tbody>
</table>
Table 26. DATABASEACTIONS work action set (continued)

<table>
<thead>
<tr>
<th>Work action</th>
<th>Work class applied to</th>
<th>Threshold type and value</th>
<th>Action</th>
</tr>
</thead>
</table>
| MAXCOSTALLOWED  | ALLACTIVITY           | Estimated SQL cost up to 1 000 000 timerons | • Stop execution  
                  |                                      |                                    | • Collect activity data           |

When these work actions are applied, up to 1000 small DML-type SQL statements (because of the SMALLDML work class) can run at a time, and up to 10 000 of these statements can be queued. Only two occurrences of the load utility can run at a time, and up to 10 occurrences can be queued. Only 100 activities that are not LOAD and are not small DML are permitted to run at a time, and only 100 of these activities can be queued at a time. In all situations, if the queued threshold is violated, the database activity is not permitted to run and an error message is returned.

In addition, the MAXCOSTALLOWED work action is applied to the ALLACTIVITY work class. This means that a database activity with an estimated cost (that is, DML and XQueries statements) of more than 1 000 000 timerons is not permitted to run. Although the MAXCOSTALLOWED work action is applied to the ALLACTIVITY work class, this work action only affects database activities that have an estimated cost greater than 1 000 000 timerons. This work action does not affect activities that do not have an estimated cost, such as DDL.
Chapter 3. Activities management

Once you have identified the work running on your data server, you are ready to actively manage this work by assigning resources and imposing controls.

Resource assignment with service classes

A service class defines an execution environment in which work can run. This execution environment allocates available resources and can include thresholds that determine how work is permitted to run.

All work runs in a service class and you use workloads to assign work to service superclasses, or you assign work to service subclasses in a service superclass by using workloads, the REMAP ACTIVITY threshold action, or the MAP ACTIVITY work action. When you define a workload, you indicate the service class where work associated with that workload runs. By default, a default user workload also exists (SYSDEFAULTUSERWORKLOAD) that maps work to the default user service class (SYSDEFAULTUSERCLASS), so that any work not explicitly mapped to a user defined service class using a user defined workload will run in the default user service class.

Without service classes, requests cannot be organized into recognizable, logical groupings, as is shown in the following figure.

Figure 9. Unorganized work

You can create different service superclasses to provide the execution environment for different types of work, then assign the applicable requests to the service superclasses. Assume that you have applications from two separate lines of business, finance and inventory. Each line of business would have its own applications to fulfill its responsibilities to the organization. You can organize the requests into categories that make sense for your workload management objectives. In the following figure, different service superclasses are assigned to different lines of business.
In the previous figure, the activities in both service superclasses are further subdivided. The service class provides a two-tier hierarchy: a service superclass and service subclasses underneath. This hierarchy permits for a more complex division of execution environment and better emulates a real-world model. Unless specified otherwise, service subclasses inherit characteristics from the service superclass. Use the service subclasses to further subdivide work in the service superclass.

**Prioritization and resource control**

When you create or alter a service class object, you can define a number of resource controls:

**Table 27. Resource control afforded by service classes**

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefetch priority</td>
<td>This control assigns a priority to the prefetch requests, which affects the order in which they are addressed by the data server.</td>
</tr>
<tr>
<td>Buffer pool priority</td>
<td>This control assigns a buffer pool priority to service classes which affects how likely pages fetched by activities in a service class are to be swapped out.</td>
</tr>
<tr>
<td>Outbound correlator</td>
<td>This control permits a workload to have some of its resources controlled by an operating system workload manager like AIX Workload Manager or Linux workload management. The tag flows through the agent to the external workload manager and maps to a resource group defined with the manager. When DB2 workload management is used in conjunction with an operating system workload manager, additional controls are available. With AIX Workload Manager, you can control the amount of processor resource allocated to each service class by setting a minimum, maximum, or relative share of processor resource for each service class. With Linux workload management, you can control the amount of CPU resource by setting shares for each service class relative to the Linux default class. Note: This control cannot be set when agent priority is in use.</td>
</tr>
</tbody>
</table>
Service subclasses

Although the service superclass is the highest tier for work, activities run only in service subclasses. Each service superclass has a default service subclass defined to run activities that you do not assign to an explicitly defined subclass. This default subclass is created when the service superclass is created. You can create additional subclasses in a service class as you require them to further isolate work. Except for histograms and the COLLECT ACTIVITY DATA, COLLECT AGGREGATE ACTIVITY DATA and COLLECT AGGREGATE REQUEST DATA options, a service subclass inherits the attributes of its service superclass, unless otherwise specified. The resources of the superclass are shared by all subclasses in it.

You can define only a single level of subclasses (that is, you cannot define a subclass under another subclass, only under a service superclass).

The following figure is an example of a custom DB2 workload management configuration using workloads and service classes:
As user requests enter the data server, they are identified as belonging to a given workload and assigned to a service superclass or subclass. There are also system requests (for example, prefetches) that run under a special default system service class (SYSDEFAULTSYSTEMCLASS) and DB2-driven maintenance requests (such as an automatic RUNSTATS from the health monitor) that run under a default maintenance service class (SYSDEFAULTMAINTENANCECLASS).

You can view your service classes by querying the SYSCAT.SERVICECLASSES catalog view.

**Default service superclasses and subclasses**

Each new database or upgraded database has three predefined default service superclasses: the default user class, the default maintenance class, and the default system class.

You cannot disable or drop any of the default service superclasses.

*Figure 11. A custom DB2 workload management configuration using workloads and service classes*
All of the default service superclasses are created with one default service subclass. You cannot create additional service subclasses for the default service superclasses. The default service subclass is always created with the name SYSDEFAULTSUBCLASS, as follows:

SYSDEFAULTUSERCLASS
    SYSDEFAULTSUBCLASS

SYSDEFAULTSYSTEMCLASS
    SYSDEFAULTSUBCLASS

SYSDEFAULTMAINTENANCECLASS
    SYSDEFAULTSUBCLASS

Figure 12. Two-tier service class hierarchy

All work issued by connections to a default service superclass are processed in the default service subclass of that service superclass.

Default service superclasses and their default service subclasses are dropped only when the database is dropped. They cannot be dropped using the DROP SERVICE CLASS statement.

Default user service superclass (SYSDEFAULTUSERCLASS)
    By default, all user activities run in the SYSDEFAULTUSERCLASS.

Default maintenance service superclass (SYSDEFAULTMAINTENANCECLASS)
    The default maintenance service superclass tracks the internal DB2 connections that perform database maintenance and administration tasks. Connections from the DB2 asynchronous background processing (ABP) agents are mapped to this service superclass. ABP agents are internal agents that perform database maintenance tasks. Asynchronous index cleanup (AIC) is an example of an ABP-driven task. ABP agents automatically reduce their resource consumption and number of subagents when the number of user connections increases on the data server. Utilities that are issued by user connections are mapped using regular service classes. You cannot implement service class thresholds on SYSDEFAULTMAINTENANCECLASS.

    The internal connections tracked by the default maintenance service superclass include:
    • ABP connections (including AIC)
    • Health monitor initiated backup
    • Health monitor initiated RUNSTATS
    • Health monitor initiated REORG

Default system service superclass (SYSDEFAULTSYSTEMCLASS)
    The default system service superclass tracks internal DB2 connections and threads that perform system-level tasks. You cannot define service subclasses for this service superclass, nor can you associate any workloads or work actions with it. In addition, you cannot implement service class
thresholds on SYSDEFAULTSYSTEMCLASS. The DB2 threads and connections tracked by the default system service superclass include:

- ABP daemon
- Self Tuning Memory Manager (STMM)
- Prefetcher engine dispatchable units (EDUs) (db2pfchr)
- Page cleaner EDUs (db2pclnr)
- Log reader EDUs (db2loggr)
- Log writer EDUs (db2loggw)
- Log file reader EDUs (db2lfr)
- Deadlock detector EDUs (db2dlock)
- Event monitors (db2evm)
- Event monitor fast writers (db2fw)
- Connections performing system level tasks

Activity-to-service class mapping

All database connections are assigned to a workload at the beginning of the first unit of work. When a workload occurrence is started, all activities running under that workload occurrence are mapped to service classes based on the service class name specified in the workload definition.

The data server assigns a connection to a workload definition if the connection meets the criteria defined for that workload definition. For example, you can set up a workload management configuration so that all connections from application A belong to the workload definition Alpha, while all connections from application B belong to the workload definition Beta.

If the workload occurrence is assigned to a service superclass, activities submitted for that workload occurrence can be reassigned to a user-defined service subclass in that service superclass using a work action set.

You can use the workload to map activities from a connection to a service superclass by specifying the SERVICE CLASS keyword of the CREATE WORKLOAD statement. Assuming that no work class or work action applies to the activity, the activity is run in the default service subclass of the service superclass. You can also use a workload to map activities from a connection to a service subclass in the service superclass by specifying the UNDER keyword for the SERVICE CLASS keyword of the CREATE WORKLOAD statement. In this situation, the connection still belongs to the service superclass, but all activities issued from that connection are automatically mapped to the service subclass specified in the workload definition.

After an activity has been mapped to a service subclass and has begun executing, it stays in that service subclass, unless you remap it to another service subclass (with a threshold). Remapping is the process by which you can change the resource assignments for an activity through a different activity-to-service subclass mapping. Both the source and the target service subclasses must exist under the same superclass. After the activity is remapped, it continues to execute in the new service subclass.

The following figure shows the relationship between connections, workloads, and service superclasses. Connections that meet the definition of workload A are mapped to service superclass 1; connections that meet the definition of workloads B or C are mapped to service superclass 2; connections that meet the definition of
workload D are mapped to the SYSDEFAULTUSERCLASS service superclass.

**Further differentiating between activities**

If you have a more complex DB2 workload manager configuration, you might want to handle activities differently based on either the activity type or some other activity attribute. For example, you might want to do one of the following actions:

- Put DML in a different service subclass than DDL.
- Put all read-type queries with an estimated cost of less than 100 timerons in a different service subclass than all the other read-type queries.

In a more complex configuration you can set up the workload to map activities from the connection to the service superclass. Then, using work actions (contained in a work action set that is applied to the service superclass), you can remap activities, based on their type or attribute, to specific service subclasses in a service superclass.
Specifically, you could apply a work action set that contains a MAP ACTIVITY work action to the service superclass. All activities that are both mapped to the service superclass and match a work class to which a MAP ACTIVITY work action is associated are mapped to the service subclass specified by the work action.

If a workload definition associates itself with a service subclass, any activities that are submitted through that workload are not affected by a work action set that is assigned to the parent service superclass.

- An activity can be mapped to one service subclass in a service superclass by a workload.
- A work action that maps the activity to a different service subclass in the same service superclass also applies to the activity.

If an activity is not mapped to a service subclass through a workload or a work action, the activity is mapped to the default subclass (SYSDEFAULTSUBCLASS) of the service superclass for that activity.

When database activities have been mapped to their respective service superclasses and service subclasses, you can implement controls on all the activities in a particular service class. Statistics are available at the service-class level that you can use to monitor database activities in that service class.

The following figure shows requests to the database being mapped to a service superclass or service subclass through workloads. For information on how work actions are used to map activities to a service subclass, see “Work actions and work action sets” on page 94.
Connections that do not map to a user-defined workload definition are mapped to the default user workload definition, SYSDEFAULTUSERWORKLOAD. By default, connections from the default workload definition (SYSDEFAULTUSERWORKLOAD) are mapped to the SYSDEFAULTUSERCLASS service superclass, which is the default service superclass for user requests. You can alter the SYSDEFAULTUSERWORKLOAD workload so that it maps to a different service class. Internal DB2 maintenance connections are mapped to the SYSDEFAULTMAINTENANCECLASS, which is the default service superclass for maintenance requests. Internal system entities and connections are mapped to SYSDEFAULTSYSTEMCLASS, which is the default service superclass for internal DB2 connections and threads that perform system-level tasks.

Figure 14. Database connections being mapped to a service superclass
**Prefetch priority of service classes**

Prefetchers retrieve data from disk and store this data in buffer pools so that it can be quickly accessed by agents. Each service superclass and subclass can be assigned to have a different prefetch priority.

Agents send read-ahead requests to the database prefetch queue. The prefetchers take these read-ahead requests from the queue, then retrieve the data into the buffer pools. When an agent requires specific data, it first checks the buffer pools to see if the data is available. If not, the agent retrieves the data from disk. Prefetchers perform expensive disk I/O operations, which frees agents to perform computational work in parallel.

Any connection routed to a service class has its prefetch requests processed according to the prefetch priority assigned for the service class. Each service class can be associated with one of the three prefetch priorities: high, medium, or low. You specify the prefetch priority of a service class with the PREFETCH PRIORITY keyword on either the CREATE or ALTER SERVICE CLASS statement.

Specifying DEFAULT for a service superclass sets a medium prefetch priority for the service superclass. You can specify a different prefetch priority for any service subclass in the service superclass, but if you use the default prefetch priority for the service subclass, the service subclass inherits its prefetch priority setting from its service superclass.

High-priority prefetch requests are processed before medium-priority prefetch requests, which, in turn, are processed before low-priority prefetch requests. Prefetch priority affects the order in which prefetch requests are processed, but it does not affect the speed at which they are processed.

**Buffer pool priority of service classes**

Setting the buffer pool priority of service classes allows you to influence the proportion of pages in the buffer pool that may be occupied by activities in a given service class, which can improve the throughput and performance of activities in that service class.

You can associate each DB2 service class with a relative buffer pool priority, which controls how likely pages fetched into the buffer pool by activities in the service class are to be swapped out. Increasing the buffer pool priority potentially increases the proportion of pages in use by agents of a particular service class.

If you do not specify a buffer pool priority, or if you specify BUFFERPOOL PRIORITY DEFAULT, the buffer pool priority assigned to the service class is DEFAULT. For service superclasses, DEFAULT maps to a value of LOW; for service subclasses, DEFAULT maps to the value of the buffer pool priority of the parent service superclass. All default subclasses have a buffer pool priority of DEFAULT, which cannot be changed.

When upgrading from an earlier DB2 version, the buffer pool priority of existing service classes is set to DEFAULT.

**Realizing the benefits of setting buffer pool priority**

You are more likely to realize a performance advantage with setting the buffer pool priority for a service class if there is a reasonable amount of contention on the buffer pool. Buffer pool contention demonstrated by an overall hit ratio of 85% or
less is likely to see the most benefit. If the overall hit ratio exceeds 90%, there is likely not substantial buffer pool contention to begin with, and setting buffer pool priority will yield less or little benefit in most cases. What benefits you realize are dependent on the type of workload your data server runs.

For some workloads, setting buffer pool priority is more effective if you also turn on proactive page cleaning. This is because buffer pool priority settings are effective only for non-dirty pages and proactive page cleaning is more aggressive about writing out dirty pages to disk. Note that you should turn on proactive page cleaning only if it yields a performance benefit.

If you use asynchronous page cleaning (also known as classic page cleaning), setting the `chngpgs_thresh` database configuration parameter to a lower value will likely yield the same effect of making your buffer pool priority settings more effective, because a low value for this parameter also ensures that there are enough clean pages in the buffer pool.

It is possible that the positive effects of setting buffer pool priority can be surpassed by the effects of prefetching, with or without setting prefetch priority, if there is a reasonable amount of prefetching taking place. For example, if you define a service class with high buffer pool priority where there is only little prefetching, the effective advantage of this buffer pool priority setting might be small when compared to a service class with low buffer pool priority but where activities perform a significant amount of prefetching. Due to the benefits of prefetching, the activities in the service class with low buffer pool priority might even outperform the activities in the high buffer pool priority service class. However, setting buffer pool priority can still supplement your workload management strategy under these circumstances, and you should use it.

**States of connections and activities in a service class**

Service classes collect connection statistics for each service class. You can see which connections and activities are in a service class, and the state of either the connection or activity.

**States of a connection**

Following are the possible states of a connection in a service class:

**CONNECTED**
- The connection successfully connected to the database but is not yet associated with its workload and service superclass.

**DECOUPLED**
- The connection does not have a coordinator agent assigned (concentrator case).

**DISCONNECTPEND**
- The connection is disconnecting from the database.

**FORCED**
- The connection has been forced.

**INTERRUPTED**
- The connection has been interrupted.

**MAPPED**
- The connection is mapped to a workload and has joined a service superclass. The connection can now submit activities for execution.
The connection coordinator agent is queued by a DB2 connection or activity threshold that offers queuing, such as the CONCURRENTDBCOORDACTIVITIES threshold. In a multimember database environment, this state might indicate that the coordinator agent has made an RPC to another member to obtain threshold tickets and has not yet received a response.

The connection is attempting to join a service class that has reached its connections threshold. The connection is queued to join the service class. When the service class is not violating its connections threshold, the connection will join the service class. A connection in the transient state cannot submit activities for execution.

The connection received a connect reset from the client or is being terminated because of a force or an error condition.

The connection is processing a request.

The connection is waiting for a request from the client.

Following are the possible states of an activity in a service class:

If you cancel an activity that has no agent actively working on a request for the activity, the activity is placed in the CANCEL_PENDING state and is cancelled on the next request that is received.

The activity is executing.

There is no agent actively processing a request for the activity.

The activity was created and is being prepared for execution.

The activity cannot be executed because of a concurrency threshold at the database or service class level. The activity is queued until it is permitted to execute.

The activity is being terminated.

The state of the activity is unknown.

Service classes are used for monitoring and controlling objects at the database level. However, not all DB2 entities work directly in a database.

Because service classes work in a database and are stored in the catalog tables of the database, entities that do not work in a database cannot be managed by service classes. Instance-level entities, such as the system controller and the health monitor daemons, work at the instance level and are not directly associated with any
database. Agents that perform instance attachments and gateway connections are
not managed by service classes either. Because instance attachment agents and
gateway agents do not work in a database, they are not managed by service
classes.

The following list is a partial list of entities that do not work within a database and
are not managed by service classes:
- DB2 system controllers (db2sysc)
- IPC listeners (db2ipcc)
- TCP listeners (db2tpcm)
- FCM daemons (db2fcms, db2fcmr)
- DB2 resynchronization agents (db2resync)
- Idle agents (agents with no database association)
- Instance attachment agents
- Gateway agents
- All other instance-level EDUs

Creating a service class

You create service superclasses and service subclasses under them using the DDL
statement CREATE SERVICE CLASS.

Before you begin

To create a service class, you require WLMADM or DBADM authority.

Also see the following topics for other prerequisites:
- “Workload management DDL statements” on page 18
- Appendix A, “General naming rules,” on page 533

Procedure

To create a service class:
1. Specify one or more of the following properties for the service class on the
CREATE SERVICE CLASS statement:
   - Specify the name of the service class:

     Note: Once set, the name of a service class cannot be changed.
     - If you are creating a service superclass, the name must be unique among
       all service superclasses in the database.
     - When a service superclass is created, its associated default service subclass
       is automatically created. Only after you have created a service superclass
       can you create other service subclasses under it.
     - If you are creating a service subclass, the name must be unique among all
       service subclasses in the service superclass. A service subclass cannot have
       the same name as its service superclass.
     - If you are creating a service subclass, specify the name of the parent service
       superclass. After a service subclass is created under a service superclass, it
       cannot be associated with a different service superclass.
     - Specify the buffer pool priority for the service class which affects how likely
       pages fetched by activities in the service class are to be swapped out. For
service superclasses, the DEFAULT value internally maps to LOW. Service subclasses set to DEFAULT inherit the buffer pool priority from their parent superclasses.

- Specify the prefetch priority. You can specify the priority with which agents in the service class can submit their prefetch requests. Depending on the value specified, the prefetch requests are routed to the high, medium, or low priority prefetch queues. The default prefetch priority is medium.

- Specify the outbound correlator string if you want to associate the DB2 service class with an AIX class or a Linux class. A null value indicates no operating system workload manager association.
  
  If the outbound correlator is set, all threads in the DB2 service class are associated with the operating system workload manager by using the outbound correlator when the next activity begins.

  If the outbound correlator is set to NONE for a service subclass and the outbound correlator is specified for the associated service superclass, the service subclass inherits the outbound correlator specified for its service superclass.

- Specify the activity data to collect. When activity data collection is enabled, information about an activity is sent from the coordinator member to the applicable event monitor at the end of the activity. You can write data to the event monitor that includes information about the statement that was run, its compilation environment, and any applicable input data values. You can also specify that no activity data is collected. By default, no activity data is collected.

- Collected aggregate activity information. The aggregate activity information used for the service class only changes after the CREATE SERVICE CLASS operation is committed.

- The type of request metrics to collect for requests submitted by a connection that is associated with the service superclass you specify. By default, basic metrics are always collected for activities associated with the workload.

- The histogram templates that the service subclass should use as templates for its histograms. The histogram templates specified are reflected in the SYSCAT.HISTOGRAMTEMPLATEUSE view. For more information about histograms and histogram templates, see “Histograms in workload management” on page 258.

- Specify whether the service class is enabled or disabled.
  
  - If a service class is created as enabled (the default), connections and activities can be mapped to the service class. If a service class is created as disabled, new connections and activities mapped to it are rejected.
  
  - If you create a service superclass as disabled, all service subclasses that you associate with this service superclass behave as though they are disabled, even though they might be displayed as being enabled when you query the SYSCAT.SERVICECLASSES view.

2. Commit your changes. When you commit your changes the service class is added to the SYSCAT.SERVICECLASSES view.

**Altering a service class**

If you want to change a service class definition, use the ALTER SERVICE CLASS statement.
**Before you begin**

To alter a service class, you require SQLADM, WLMADM, or DBADM authority. To specify any clause other than a COLLECT clause, the authorization ID must include WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

**About this task**

Activities that have already acquired resources and are running are usually not affected by the ALTER statement. These activities will hold their resources and run until completion. (Note that changes to the CPU shares or CPU limits of a service class that are made through the ALTER SERVICE CLASS statement take effect immediately, even in the case of activities that are already running.) However, if a subagent request is sent to a remote member during the ALTER SERVICE CLASS operation, the service class definition seen by the coordinator agent and the subagent can differ. Consider the following example in which the prefetch priority for the service class is initially set to MEDIUM:

<table>
<thead>
<tr>
<th>Event order</th>
<th>Connection 1</th>
<th>Connection 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Coordinating agent sends a request to remote member (prefetch priority of service class was previously set to MEDIUM)</td>
<td>ALTER SERVICE CLASS issued; set prefetch priority to HIGH</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>COMMIT is issued (the altered service class property is committed at the catalog member and loaded to memory at all database members)</td>
</tr>
<tr>
<td>3</td>
<td>Remote subagent receives the request. At this time, the subagent sets the new prefetch priority of HIGH for the service class definition</td>
<td></td>
</tr>
</tbody>
</table>

This situation described in the previous table is temporary, and only affects connections that issue subagent requests during the ALTER SERVICE CLASS operation. All new connections will see the updated service class definition with the prefetch priority of HIGH.

**Procedure**

To alter a service class:

1. Specify one or more of the following properties for the service class on the ALTER SERVICE CLASS statement:
Specify whether the service class is enabled or disabled. If you change a
service class from enabled to disabled, existing connections or activities
remain with the service class and continue to use previously allocated
resources until complete. You can disable a service class if the work coming
to the service class is overwhelming the system, or you want to reject all
work coming to the service class.

When a service superclass is disabled the following happens:

a. The service superclass is disabled.
b. Its service subclasses are disabled.

The service subclasses are only disabled while their service superclass is
disabled. When the service superclass is enabled, the service subclasses
return to their previous states as defined in the catalog table.

When a service subclass is disabled, its service superclass is not affected, nor
other service subclasses associated with the service superclass.

You cannot explicitly disable a default service subclass. To prevent new
requests from running under a default service subclass, you must disable the
associated service superclass.

• Specify the prefetch priority. You can specify the priority with which agents
  in the service class can submit their prefetch requests. Depending on the
  value specified, the prefetch requests are routed to the high, medium, or low
  priority prefetch queues. The default prefetch priority is medium. If the
  prefetch priority is altered after a prefetch request is submitted, the request
  will not change its priority.

• Specify the buffer pool priority for the service class which affects how likely
  pages fetched by activities in the service class are to be swapped out. For
  service superclasses, the DEFAULT value internally maps to LOW. Service
  subclasses set to DEFAULT inherit the buffer pool priority from their parent
  superclasses.

• Specify the outbound correlator string if you want to associate the DB2
  service class with an AIX class or a Linux class. A null value indicates no
  operating system workload manager association.

  If the outbound correlator is changed from a non-null value to a null value,
  all threads in the DB2 service class will disassociate with the operating
  system workload manager when the next activity begins.

  If the outbound correlator is set to NONE for a service subclass and the
  outbound correlator is specified for the associated service superclass, the
  service subclass inherits the outbound correlator specified for its service
  superclass.

• Specify the adding or dropping of a system temporary table space to or from
  the list of preferred temporary table spaces for the specified service subclass.

• Specify the activity data to collect. When activity data collection is enabled,
  information about an activity is sent from the coordinator member to the
  applicable event monitor at the end of the activity. You can write data to the
  event monitor that includes information about the statement that was run, its
  compilation environment, and any applicable input data values. You can also
  specify that no activity data is collected. By default, no activity data is
  collected.

• Collected aggregate activity information. The aggregate activity information
  used for the service class only changes after the ALTER SERVICE CLASS
  operation is committed.

• The monitoring request metrics collection level for requests submitted by
  connections mapped to a subclass under the specified service superclass.
Note that the effective collection setting for requests running under a service superclass is the combination of both the service class collection level and the mon_req_metrics database configuration parameter.

- Whether to alter the histogram templates used by a service subclass that has enabled aggregate activity data collection using COLLECT AGGREGATE ACTIVITY DATA or aggregate request data collection using COLLECT AGGREGATE REQUEST DATA. Updating the histogram templates used by a service subclass will update the corresponding rows in the SYSCAT.HISTOGRAMTEMPLATEUSE view which displays the histogram templates referenced by a service class or work action. For more information about histograms and histogram templates, see “Histograms in workload management” on page 258.

2. Commit your changes. When you commit your changes the service class is updated in the SYSCAT.SERVICECLASSES view.

Dropping a service class

You drop service classes using the DDL statement DROP SERVICE CLASS.

Before you begin

To drop a service class, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

You cannot drop the default service superclasses (SYSDEFAULTUSERCLASS, SYSDEFAULTMAINTENANCECLASS, and SYSDEFAULTSYSTEMCLASS) or their associated service subclasses. The only way to drop the default service superclasses and their associated service subclasses is to drop the database.

A service class you defined cannot be dropped if any of the following conditions apply:

- It is enabled
- It contains user-defined service subclasses
- It is referenced by any workload, work action, or threshold
- It is still referenced by a workload occurrence
- Any connection or activity is currently mapped to the service class
- If the service class is set as the target of a REMAP ACTIVITY action.

Procedure

To drop a service class:

1. Change workloads to not point to the service class, or disable the workload definitions. Alternatively, use the DROP WORKLOAD statement to drop all workloads that are associated with the service class. Issue a COMMIT statement after dropping each workload.

Activities already running under the service class will continue to run. You can list agents that are currently mapped to a service class by using the WLM_GET_SERVICE_CLASS_AGENTS table function. If you do not want these activities to complete, you can use the application identifier returned by the table function and use the FORCE APPLICATION command to force these applications off the database.
2. Drop all applicable work actions that are associated with the service class you want to drop:
   - If you are dropping a service superclass and a work action set is associated with it, disable and drop that work action set. Use the ALTER WORK ACTION SET statement to disable a work action set, and the DROP WORK ACTION SET statement to drop the work action set. Issue a commit statement after dropping the work action set.
   - If you are dropping a service subclass and a work action maps to that service subclass, drop the work action using the DROP WORK ACTION clause of the ALTER WORK ACTION SET statement. Alternatively, drop the work action set that contains the work action that maps to the service subclass by using the DROP WORK ACTION SET statement. Issue a COMMIT statement after dropping each work action, or after dropping the work action set.

3. Disable and drop all thresholds associated with the service class that you want to drop. Issue a COMMIT statement after dropping each threshold. Use the ALTER THRESHOLD statement to disable a threshold, and the DROP THRESHOLD statement to drop the threshold.

4. Depending on the object you are dropping, do the following:
   - If you are dropping a service subclass, use the DROP SERVICE CLASS statement to drop the service subclass.
   - If you are dropping a service superclass, use the DROP SERVICE CLASS statement to drop all service subclasses associated with the service superclass, and issue a COMMIT statement after dropping each service subclass. Then issue the DROP SERVICE CLASS statement to drop the service superclass.

   **Note:** You cannot manually drop the default service subclass for the service superclass. The default service subclass for a service superclass is dropped when the service superclass is dropped.

5. Disable the service class by using the ALTER SERVICE CLASS statement. If you are dropping a service superclass, this action disables all service subclasses associated with the service superclass. Disabling a service class prevents any additional activities from being associated with it. After disabling the service class, issue a COMMIT statement.

6. Commit your changes. When you commit your changes, the service class is removed from the SYSCAT.SERVICECLASSES view.

**Example: Using service classes**

The following example shows how to use service classes to control database workload.

This example occurs in the fictitious International Beer Emporium. International Beer Emporium is a medium-sized business made up of five major departments: Sales, Accounting, Engineering, Testing and Production. All five departments share the same product catalog database.

**Initial implementation of a DB2 workload management solution**

The product catalog database runs well most of the time. However, sometimes users complain that their applications cannot connect to the database because the maximum number of connections has been exceeded. After upgrading to DB2 Version 9.7, Bob, the database administrator, decides to try service classes. Bob wants to know the usage patterns of the product catalog database by each of the
five departments and figure out why his database runs out of connections occasionally. Following are the steps Bob follows to set up the service classes:

1. First, Bob creates service superclasses for each of the departments (the default service subclass is also automatically created for each service superclass):
   - **SALES** is created for the Sales department:
     
     ```
     CREATE SERVICE CLASS SALES
     ```
   - **ACCOUNTING** is created for the Accounting department:
     
     ```
     CREATE SERVICE CLASS ACCOUNTING
     ```
   - **ENGINEERING** is created for the Engineering department:
     
     ```
     CREATE SERVICE CLASS ENGINEERING
     ```
   - **TESTING** is created for the Testing department:
     
     ```
     CREATE SERVICE CLASS TESTING
     ```
   - **PRODUCTION** is created for the Production department:
     
     ```
     CREATE SERVICE CLASS PRODUCTION
     ```

2. Bob creates session user groups with appropriate authorization IDs for each of the departments:
   - A session user group is created with the authorization ID SALESGRP. This group includes the authorization IDs of all users in the Sales department.
   - A session user group is created with the authorization ID ACCTNGRP. This group includes the authorization IDs of all users in the Accounting department.
   - A session user group is created with the authorization ID ENGINGRP. This group includes the authorization IDs of all users in the Engineering department.
   - A session user group is created with the authorization ID TESTGRP. This group includes the authorization IDs of all users in the Testing department.
   - A session user group is created with the authorization ID PRODGRP. This group includes the authorization IDs of all users in the Production department.

3. Bob creates workloads to map connections from each group to the associated service class:
   - Workload **WL_SALES** is created with its session user group set to SALESGRP. WL_SALES maps its connections to the service superclass SALES:
     
     ```
     CREATE WORKLOAD WL_SALES SESSION_USER GROUP ('SALESGRP')
     SERVICE CLASS SALES
     ```
   - Workload **WL_ACCOUNTING** is created with its session user group set to ACCTNGRP. WL_ACCOUNTING maps its connections to the service superclass ACCOUNTING:
     
     ```
     CREATE WORKLOAD WL_ACCOUNTING SESSION_USER GROUP ('ACCTNGRP')
     SERVICE CLASS ACCOUNTING
     ```
   - Workload **WL_ENGINEERING** is created with its session user group set to ENGINGRP. WLENGINEERING maps its connections to service class ENGINEERING:
     
     ```
     CREATE WORKLOAD WL_ENGINEERING SESSION_USER GROUP ('ENGINGRP')
     SERVICE CLASS ENGINEERING
     ```
   - Workload **WL_TEST** is created with its session user group set to TESTGRP. WL_TEST maps its connections to service class TESTING:
     
     ```
     CREATE WORKLOAD WL_TEST SESSION_USER GROUP ('TESTGRP')
     SERVICE CLASS TESTING
     ```
Workload WL_PRODUCTION is created with its session user group set to PRODGRP. WL_PRODUCTION maps its connections to service class PRODUCTION:

```
CREATE WORKLOAD WL_PRODUCTION SESSION_USER_GROUP ('PRODGRP')
SERVICE_CLASS PRODUCTION
```

Bob uses the default service class and workload settings. He wants to observe the database usage patterns before placing any controls on the service classes. The resulting service superclass definitions are as follows:

- SALES
- ACCOUNTING
- ENGINEERING
- TESTING
- PRODUCTION
- SYSDEFAULTUSERCLASS
- SYSDEFAULTMAINTENANCECLASS
- SYSDEFAULTSYSTEMCLASS

With a DB2 workload management solution implemented as described previously, work from each department is routed to its own service superclass. Work from departments not specifically accounted for is mapped to the SYSDEFAULTUSERCLASS default service superclass. Using this configuration, Bob can monitor the work in each of the service classes to determine the database usage pattern of the departments.

**First refinement of the DB2 workload management implementation**

Following the most recent connection spike, Bob queries service superclass statistics using the WLM_GET_SERVICE_SUPERCLASS_STATS table function and examines the connection high-water mark value for each service superclass. Bob discovers that the connection high-water mark for all departments except Testing is close to 100. However, the statistic for the Testing department shows that at one time, the test team established over 800 connections.

Once a month, the Testing department performs its monthly intensive product testing. At this time, the department establishes up to 1000 concurrent connections. Because the database manager configuration parameter `max_connections` is set to 1000, the Testing department uses most of the available connections to the database. When the system has 1000 connections, all subsequent connections are rejected.

Because of memory constraints on the system, the `max_connections` and `maxagents` configuration values cannot be increased on the data server to permit more connections.

To prevent the Testing department from using all the connections, Bob decides to limit the number of connections from the Testing department and ensure that each of the other four departments can obtain sufficient connections to the database to meet their business objectives.

The other four departments ordinarily do not require more than 150 concurrent connections each. In addition, Bob also notices that the default user, default maintenance, and default system service superclasses rarely contain any
connections, so he decides that 100 connections should be sufficient for these default service superclasses. After 700 connections (600 for the four departments and 100 for the default classes) are allocated from the `max_connections` pool of 1,000 available connections, 300 connections are available for the Testing department. By limiting the Testing department to a maximum of 300 connections, users from other departments should not have their connection requests rejected.

To limit the Testing group to a maximum of 300 concurrent connections, Bob creates a `MAXSERVICECLASSCONNECTIONS` threshold of 300 for the `TESTING` service class.

```
CREATE THRESHOLD MAXSERVICECLASSCONNECTIONS FOR SERVICE CLASS TESTING ACTIVITIES
ENFORCEMENT DATABASE PARTITION
WHEN TOTALSCMEMBERCONNECTIONS > 300 STOP EXECUTION
```

After implementing this change, the DB2 workload management configuration is as follows:

**Table 29. Configuration after adding threshold for the TESTING service superclass**

<table>
<thead>
<tr>
<th>Service class</th>
<th>MAXSERVICECLASSCONNECTIONS threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES</td>
<td>N/A</td>
</tr>
<tr>
<td>ACCOUNTING</td>
<td>N/A</td>
</tr>
<tr>
<td>ENGINEERING</td>
<td>N/A</td>
</tr>
<tr>
<td>TESTING</td>
<td>N/A</td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>300</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>N/A</td>
</tr>
<tr>
<td>SYSDEFAULTMAINTENANCECLASS</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Because the `TESTING` service class can contain a maximum of only 300 concurrent connections, all connection requests above this threshold are rejected. A `MAXSERVICECLASSCONNECTIONS` threshold is not applied on the other service classes, so these service classes share the remaining 700 available connections to the data server. Because there is no contention for connections among these service classes, Bob does not place connection thresholds on them.

**Second refinement of the DB2 workload management implementation**

Although connections from the Sales, Accounting, Engineering, and Production departments are no longer being rejected, users from these departments still complain about poor performance when the Testing department performs intensive product testing. Bob examines the queries that the Testing department runs during its product test cycle and discovers that the queries contain complicated joins that involve large amounts of data. These queries generate considerable prefetch activity, which prevents connections from other departments having their prefetch requests processed. Bob decides to lower the prefetch priority of the connections from the Testing department and alters the `TESTING` service class to set its prefetch priority to `LOW`:

```
ALTER SERVICE CLASS TESTING PREFETCH PRIORITY LOW
```

The DB2 workload management configuration is as follows:
Table 30. Configuration after changing prefetch priority for the TESTING service superclass

<table>
<thead>
<tr>
<th>Service class</th>
<th>MAXSERVICECLASSCONNECTIONS threshold</th>
<th>Prefetch priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>ACCOUNTING</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>ENGINEERING</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>TESTING</td>
<td>300</td>
<td>LOW</td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>SYSDEFAULTMAINTENANCECLASS</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>

Setting the prefetch priority of the TESTING service class to LOW causes prefetch requests from connections issued from the Testing department to be serviced only after all prefetch requests from the other departments are processed. This change increases the query throughput of the other departments and decreases the throughput of the Testing department during its product testing phase.

Third refinement of the DB2 workload management implementation

After the prefetch problem is resolved, the Engineering department tells Bob that it needs a few connections for an experimental application called Brewmeister. Because the application is experimental, Bob wants to ensure that it does not consume too many database connections and that queries from the application will not compete for prefetchers when the system is busy. To accomplish these objectives, he creates a new service subclass under the ENGINEERING service superclass for the experimental application and a workload to map connections from the application to the new service subclass. Bob updates the service class and workloads as follows:

- Service subclass EXPERIMENT is created under the service superclass ENGINEERING:
  
  CREATE SERVICE CLASS EXPERIMENT UNDER ENGINEERING

- Threshold MAXSERVICECLASSCONNECTIONS of 50 is created for the service subclass EXPERIMENT:
  
  CREATE THRESHOLD MAXSERVICECLASSCONNECTIONS FOR SERVICE CLASS EXPERIMENT UNDER ENGINEERING ACTIVITIES
  
  ENFORCEMENT DATABASE WHEN TOTALMEMBERCONNECTIONS > 50 STOP EXECUTION

- Workload WL_EXPERIMENT is created to map connections from the application BREWMEISTER to the service subclass EXPERIMENT:
  
  CREATE WORKLOAD WL_EXPERIMENT APPLNAME ('BREWMEISTER') SERVICE CLASS EXPERIMENT UNDER ENGINEERING

- The prefetch priority for the EXPERIMENT service subclass is set to LOW:
  
  ALTER SERVICE CLASS EXPERIMENT UNDER ENGINEERING PREFETCH PRIORITY LOW

The DB2 workload management configuration is as follows:

Table 31. Configuration with EXPERIMENT service subclass

<table>
<thead>
<tr>
<th>Service class</th>
<th>MAXSERVICECLASSCONNECTIONS threshold</th>
<th>Prefetch priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>
Table 31. Configuration with EXPERIMENT service subclass (continued)

<table>
<thead>
<tr>
<th>Service class</th>
<th>MAXSERVICECLASSCONNECTIONS threshold</th>
<th>Prefetch priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTING</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>ENGINEERING</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>EXPERIMENT</td>
<td>50</td>
<td>LOW</td>
</tr>
<tr>
<td>TESTING</td>
<td>300</td>
<td>LOW</td>
</tr>
<tr>
<td>PRODUCTION</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>SYSDEFAULTMaintenanceCLASS</td>
<td>N/A</td>
<td>DEFAULT</td>
</tr>
</tbody>
</table>

With this configuration, the BREWMEISTER application can only maintain 50 concurrent connections to the database. In addition, prefetch requests from this application are sent to the low priority prefetch queue. The Engineering department can now safely experiment with the application, knowing that it cannot accidentally overwhelm the database system.

**Example: Analyzing a service class-related system slowdown**

If you notice a system slowdown (for example, some applications take much longer than expected to complete) and are unsure whether the problem is related to the configuration of the service classes, you can use table function data to investigate and, if necessary, correct the problem.

First, obtain a high-level overview of what is occurring in the service classes. This high-level overview should include the average activity lifetime, the number of activities that completed normally rather than abnormally, and the high watermark for concurrent coordinator activities in the system. To obtain this information, you can create a general query with aggregation across service classes and database members by using the data obtained from the table function `WLM_GET_SERVICE_SUBCLASS_STATS`. Set the first and second arguments to empty strings and the third argument to -2 (a wildcard character) to indicate that data is to be gathered for all service classes on all database members. Your query might resemble the following one:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(SUM(COORD_ACT_COMPLETED_TOTAL)),1,13) AS ACTSCOMPLETED,
       SUBSTR(CHAR(SUM(COORD_ACT_ABORTED_TOTAL)),1,11) AS ACTSABORTED,
       SUBSTR(CHAR(MAX(CONCURRENT_Act_TpO)),1,6) AS ACTSHW,
       CAST(CASE WHEN SUM(COORD_ACT_COMPLETED_TOTAL) = 0 THEN 0
            ELSE SUM(COORD_ACT_COMPLETED_TOTAL * COORD_ACT_LIFETIME_AVG)
                 / SUM(COORD_ACT_COMPLETED_TOTAL) END / 1000 AS DECIMAL(9,3))
       AS ACTAVGLIFETIME
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS ('', '', -2)) AS SCSTATS
GROUP BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
ORDER BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
```

Assume that on previous occasions, the query reported the following results:

<table>
<thead>
<tr>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
<th>ACTSCOMPLETED</th>
<th>ACTSABORTED</th>
<th>ACTSHW</th>
<th>ACTAVGLIFETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS 8</td>
<td>0</td>
<td>1</td>
<td>3.750</td>
<td></td>
</tr>
<tr>
<td>BI_APPS</td>
<td>SYSDEFAULTSUBCLASS 4</td>
<td>0</td>
<td>1</td>
<td>5.230</td>
<td></td>
</tr>
<tr>
<td>BATCH</td>
<td>SYSDEFAULTSUBCLASS 1</td>
<td>0</td>
<td>1</td>
<td>25.600</td>
<td></td>
</tr>
</tbody>
</table>

The data returned by this query might be sufficient to show that the slowdown is occurring in the BI_APPS service class because its average activity lifetime is
significantly higher than usual. This situation could indicate that the available resources for that particular service class are becoming exhausted.

If the averages for the service classes for all database members do not isolate the problem, consider analyzing average values for each member. Aggregating the average for each member into a global average can hide large discrepancies between database members. In this situation, the assumption is that every member is being used as a coordinator member. If this assumption is incorrect, the average lifetime computed at non-coordinator members is zero.

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEM,
       CAST(COORD_ACT_LIFETIME_AVG / 1000 AS DECIMAL(9,3)) AS AVGLIFETIME
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '', -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME
```

<table>
<thead>
<tr>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
<th>MEM</th>
<th>AVGLIFETIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>3.425</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>1</td>
<td>2.752</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2</td>
<td>8.230</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>3</td>
<td>0.593</td>
</tr>
</tbody>
</table>

In this example, member 2 might be receiving more work than usual because its average activity lifetimes are much higher than those of the other database members.

Many different situations can cause a system slowdown. Use the following principles to make the best use of the information provided by the DB2 workload management table functions:

- Address large numbers of locking conflicts at the level of the application logic and environment (isolation level and so on).
- If the service class is running close to its threshold levels (the number of concurrent requests and so on), you might need to increase the thresholds.
- If the resources allotted to a service class are becoming exhausted and OUTBOUT CORRELATOR is set, the mapping to the operating system service classes might be the cause of the problem (that is, the operating system service class corresponding to the service class is not getting enough processor resources).
- Higher numbers of activities than expected might be running in the service class, which might be consuming more resources than normal. Check the number of completed activities to determine whether the amount of work being done in the service class is reasonable.
- Activities might be spending more time in queues if more activities are being submitted than expected and concurrency thresholds are defined. Check whether the average queue time for activities has increased by the same amount as the average lifetime. If they have increased by the same amount, the queues are behaving as expected; however, if the lifetime is unacceptable, consider allocating more resources to the service class and reducing the concurrency threshold.

**Example: Investigating agent usage by service class**

DB2 workload management provides the `WLM_GET_SERVICE_CLASS_AGENTS` table function, which you can use to determine the relative distribution of agents among service classes.
Situations can arise in which a data server resource, such as an agent, is overutilized by a group of users or an application. For example, assume that a group of users is using almost all of the available agents and that a user from outside this group voices a concern about that to you.

The first step to take is to determine how many agents are working for each service class. You might use a query such as the following one:

```sql
SELECT SUBSTR(AGENTS.SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(AGENTS.SERVICE_SUBCLASS_NAME,1,19) AS SUBCLASS_NAME,
       COUNT(*) AS AGENT_COUNT
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS('', '', CAST(NULL AS BIGINT), -2))
AS AGENTS
WHERE AGENT_STATE = 'ACTIVE'
GROUP BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
ORDER BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
```

<table>
<thead>
<tr>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
<th>AGENT_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>7</td>
</tr>
<tr>
<td>TEST</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>20</td>
</tr>
</tbody>
</table>

If you conclude that a particular service class is using more than its fair share of agents, you can take actions to restrict the number of activities permitted for a workload or a service class. Alternatively, you can restrict the number of connections for a service class.

**Apply controls to types of activities with work action sets**

Work action sets contain work actions that apply controls to activities of a certain type in a specific service superclass, in a specific workload, or to the database as a whole.

A work action provides an action that can be applied to a work class, which represent activities of a certain type like LOAD or READ activities. Work actions are applied to the activities that fall within the work class to which the work action was applied before the activity begins to execute. If, however, the work action is PREVENT EXECUTION, the activity will be prevented from running and no other work action will be applied to it.

If you apply a work action set to a database, there are several types of actions that you can apply to activities that fall within a work class, such as threshold definitions, prevent execution, collect activity data, and count activity. Defining a threshold for a work action is the most powerful database work action. For example, perhaps you want to prevent SQL from reading and returning more than 100 000 rows. You can define a single work class for a work action set that identifies SQL READ statements and a work action with a threshold that would stop execution if the number of rows returned is more than 100 000. For information about the possible actions, see “Work actions and the work action set domain” on page 104.

If you apply a work action set to a workload, the different types of actions that you can apply to activities include defining thresholds, preventing execution, collecting activity data and aggregate activity data, and counting the activities.

If you define the work action set for a service superclass, the different types of actions that you can apply to activities include mapping activities to a service subclass, preventing execution, collecting activity or aggregate activity data, and counting the activities. Typically, the work action maps an activity to a service...
subclass and has thresholds defined on the subclass to help manage the activity.

How work classes, work class sets, work actions, and work action sets work together and are associated with other DB2 objects

Work classes and work actions work together to apply specific actions to specific activity types. The best way to describe how this works is through an example.

The following diagram shows a high-level view of how work classes, work class sets, work actions, and work action sets work together and are associated with other DB2 objects.
In the diagram, some database activities are mapped, through workload WL1, workload WL3, and the default user workload, SYSDEFAULTUSERWORKLOAD, to the service superclass SS1. Because work action set WASDB is applied to the database, all activities that enter the database (regardless of the workload) and that are classified as either the WC_DML or WC_LOAD work class will have the work actions in the WASDB work action set applied to them. That is, activities with the DML work type are counted, and activities with the LOAD work type have activity data collected for them and written to an active event monitor (if one is available).

The work action set WASSSC1 is applied to the service superclass SS1. Any activities that are assigned to the default user workload, the WL1 workload, or the WL3 workload, and that fall under the WC_DML work class and the WC_LOAD work class will also have the WA_MAP_DML and WA_MAP_LOAD work actions applied to them, because those workloads direct work to the SS1 service superclass. That is, activities with a work type of LOAD will be mapped to the SSC1 service subclass by the WA_MAP_LOAD work action, and activities with a work type of DML will be mapped to the SSC2 service subclass by the WA_MAP_DML work action.

Figure 16. Overview of work action sets and work class sets

In the diagram, some database activities are mapped, through workload WL1, workload WL3, and the default user workload, SYSDEFAULTUSERWORKLOAD, to the service superclass SS1. Because work action set WASDB is applied to the database, all activities that enter the database (regardless of the workload) and that are classified as either the WC_DML or WC_LOAD work class will have the work actions in the WASDB work action set applied to them. That is, activities with the DML work type are counted, and activities with the LOAD work type have activity data collected for them and written to an active event monitor (if one is available).

The work action set WASSSC1 is applied to the service superclass SS1. Any activities that are assigned to the default user workload, the WL1 workload, or the WL3 workload, and that fall under the WC_DML work class and the WC_LOAD work class will also have the WA_MAP_DML and WA_MAP_LOAD work actions applied to them, because those workloads direct work to the SS1 service superclass. That is, activities with a work type of LOAD will be mapped to the SSC1 service subclass by the WA_MAP_LOAD work action, and activities with a work type of DML will be mapped to the SSC2 service subclass by the WA_MAP_DML work action.
Activities that are assigned to the WL2 workload are mapped directly to a service subclass (SSC3). When a workload maps activities directly to a service subclass, no work actions from the work action set WASSSC1 are applied to those activities. However, because WASWL2 is applied to WL2, any activities assigned to WL2 and fall under WL_LOAD will have work actions in the WASWL2 work action set applied to the. That is, LOAD activities will not be allowed to be run, due to the PREVENT EXECUTION work action.

**Work actions and work action sets**

A work action, when used in conjunction with a work class, can be used to help control specific types of activities. For example, you can apply different work actions to LOAD activities so that they are processed differently than DML. Work actions are grouped into work action sets.

**Work actions**

A work action consists of the following attributes:

- A user-supplied work action name, which must be unique in the work action set.
- The work class identifier the work action is to be applied to. You can define more than one work action for a work class, but each work action must perform a different action on that work class.
- The action that is to be applied to the database activity that matches the work class. The valid action type for a work action depends on whether the work action set that the work action belongs to is applied to a database, a workload, or a service superclass. When a work action set is applied to a database, all work entering the database is evaluated by the work action set. When a work action set is applied to a workload, all work submitted through workload occurrences associated with that workload are evaluated by that work action set. When a work action set is applied to a service superclass, all work submitted for execution directly to the service superclass is evaluated by that work action set; that is, workload definitions that specifically name the service superclass as their target service class will have their submitted work evaluated by the work action set defined on the service superclass. For example:
  - A work action set that is applied to a database or a workload can contain threshold work actions. If an activity gets assigned to a work class that has a threshold work action defined for it, the threshold is applied to that activity.
  - A work action set that is applied to a service superclass can contain a work action that maps the activity to a service subclass in the service superclass. If an activity corresponds to a specific work class in a work class set, and the work action set has a mapping work action that is defined for that work class, that activity is mapped to the service subclass specified by the work action.

For a list of the supported actions, see “Work actions and the work action set domain” on page 104.

- An object that is the target of the specified action. Depending on the action, the object can be a service subclass that the activity is mapped to, a threshold that specifies which threshold to apply to the activity, or null if the action is to prevent execution, one of the collect actions, or count activity.
- The template describing the histogram that collects statistical information about the number of milliseconds that activities associated with the work class to which this work action is assigned required to run during a specific interval. This information is only collected when the work action type is COLLECT
AGGREGATE ACTIVITY DATA (either BASE or EXTENDED). For more information on histograms and histogram templates, see “Histograms in workload management” on page 258.

- Whether or not the work action is enabled.
- An automatically generated identifier that identifies the work action.

You can create a work action by using either the WORK ACTION keyword in the CREATE WORK ACTION SET statement or the ADD keyword in the ALTER WORK ACTION SET statement. You can alter a work action by using the ALTER keyword in the ALTER WORK ACTION SET statement. You can remove a work action from a work action set by using the DROP keyword in the ALTER WORK ACTION SET statement, or by dropping the entire work action set.

You can view your work actions by querying the SYSCAT.WORKACTIONS view.

**Work action sets**

A work action set consists of the following attributes:

- A work action set name that is unique in a database.
- The name of the work class set containing one or more work classes that the group of actions is to apply to.
  
  Because the definitions of the work class sets are separate from the work action sets defined for them, you can define more than one work action set for a work class set.

  - The type of object that the work action set is associated with (database, service superclass, or workload).
  - The name of the service superclass that the actions and work class set apply to (for work action sets associated with a service superclass).
  - Whether or not the work action set is enabled.
  - User comments.
  - One or more work actions (a work action set does not have to contain any work actions).
  - An automatically generated ID that uniquely identifies the work action set.

You can create a work action set using the CREATE WORK ACTION SET statement, alter a work action set using the ALTER WORK ACTION SET statement, and drop a work action set using the DROP WORK ACTION SET statement.

You can view your work action sets by querying the SYSCAT.WORKACTIONSETS view.

When you create a work action set, you must specify the object that the work action set is to be applied to. The valid object types are the database, a workload, or a service superclass. You must also specify which work class set the work action set is to work with. This permits you to use the work classes in the work class set to identify the types of activities that you want to apply the work actions to.

If you create a work action set on a service superclass, these are important aspects to remember:

1. If you set up a workload to map its database activities directly to a service subclass, the work action set associated with that service superclass is never used for the activities issued by that workload. In other words, if a workload
maps activities directly to a service subclass, the work action set is bypassed. None of the work actions in the work action set will be applied to the activities that are mapped directly to the service subclass.

2. Any activity, not mapped to a service subclass by a work action within the work action set on the superclass, executes in the SYSDEFAULTSUBCLASS of the service superclass.

3. All work submitted by a workload associated with the superclass that is not recognized as an activity, such as a PREPARE request or RUNSTATS, executes in the SYSDEFAULTSUBCLASS of the service superclass since the work action set does not affect them.

Creating a work action
Use the CREATE WORK ACTION SET statement or the ALTER WORK ACTION SET statement to create a work action.

Before you begin
To create a work action, you require WLMADM or DBADM authority.

For additional prerequisites, see the following topics:
- “Workload management DDL statements” on page 18
- Appendix A, “General naming rules,” on page 533

When you create a work action:
- You associate a work action with a work class. The work class must already exist in the work class set that the work action set is applied to.
- If the work action is a threshold, the work action set must be defined for a workload or for the database. For the list of supported thresholds for work actions, see “Thresholds that can be used in work actions” on page 108.
- If you are creating a mapping work action, the work action set must be defined for a service superclass. The service subclass being mapped to must already exist in the service superclass this work action set is being defined for. In addition, you cannot specify the default service subclass.
- More than one work action can be applied to a single work class, but they must be different work action types. For example, you can apply a mapping work action and a collect work action to the same work class. You can, however, only apply one work action of the same type to the same work class. For example, you cannot apply more than one mapping work action to the same work class. The one exception to this is if the work action represents a threshold. You can apply more than one threshold work action to a single work class but each threshold must be of a different type.
- If you are creating a collect aggregate activity data work action, the work action set must be defined for a service superclass or a workload.

Procedure
To create a work action:
1. Use the work-action-definition keyword of the CREATE WORK ACTION SET statement, or the ADD work-action-definition keyword of the ALTER WORK ACTION SET statement. Specify one or more of the following for the work action:
   - A name for the work action. The name of the work action must be unique within the work action set.
• The name of the work class to which this work action applies. The work
class must be one of the work classes in the work class set that the work
action set is associated with. For example, to apply this work action to the
work class LARGEDML, you would specify:
ON WORK CLASS LARGEDML
• The action that is to apply to activities that match the work class for this
work action:
  – If the work action set is associated with a service superclass, you can
    specify the MAP ACTIVITY keyword so that the work action maps
    activities to a service subclass in the service superclass. By default,
    mapping work actions cause activities that are nested to be mapped to
    the same service subclass as its parent. A cursor that has been opened inside a
    routine is an example of a nested activity.
    For example, if you want the work action to map to the service subclass
    SMALLREAD, and you want all nested activities to be mapped to the same
    service subclass, you would specify:
    MAP ACTIVITY TO SMALLREAD
    You could also specify:
    MAP ACTIVITY WITH NESTED TO SMALLREAD
    If you want the work action to map to the service subclass and to not map
    nested activities to this service subclass, you would specify:
    MAP ACTIVITY WITHOUT NESTED TO SMALLREAD

    If you define the work action as WITHOUT NESTED, nested activities are
    handled according to their activity type instead of automatically being
    mapped to the same service subclass as the parent activity. For example, if
    a CALL activity is mapped to service subclass subscl, and the routine has
    an open cursor inside it, the open cursor might be mapped to a different
    service subclass if it falls under another work class that has another
    mapping work action applied to it.
• If the work action set is associated with a database or a workload, you can
  specify a WHEN keyword to indicate a threshold to apply to the activity, and
  the action to take if the activity causes the threshold to be violated. You can
  specify the following thresholds for a work action:
  – ACTIVITYTOTALTIME
  – ESTIMATEDSQLCOST
  – CONCURRENTDPROCOCORDACTIVITIES
  – CPUTIME
  – SQLROWSREAD
  – SQLTEMPSPACE
  – SQLROWSRETURNED
  If the threshold is violated, you can specify the following actions to be taken:
  – Whether activity data is to be collected about the activity that caused the
    threshold to be violated. If collected, when the activity completes
    execution, the activity data is written to an active activities event monitor.
    By default, no data about the activity is collected. If you want to collect
    data about this activity, you can collect it from the coordinator member, a
    specific member, or from all database members. You have the option of
    collecting this data with or without details about the statement and its
    compilation environment. If you want to collect details about the
    statement and compilation environment, you can also specify that the
    input data values used in the activity.
- Whether the activity that caused the threshold to be violated is to be permitted to continue running or not. By default, the activity is stopped.

For example, if you want the work action to check for DML statements that have a cost over 2,000 timerons, collect the basic data about this activity when the threshold is violated and continue to run, you would specify:

```
WHEN ESTIMATESQLCOST > 2000 COLLECT ACTIVITY DATA CONTINUE
```

- To prevent any activities that correspond to the work class defined for this work action from executing, you can use the PREVENT EXECUTION keyword.

- To count the number of database activities associated with the work class without incurring the additional overhead of another action (such as collecting data or mapping an activity), you can specify the COUNT ACTIVITY keyword.

- To collect activity data for activities that fall under the work class, specify the COLLECT ACTIVITY DATA keyword. If collected, when the activity completes execution, the activity data is written to an active activities event monitor. By default, no data about the activity is collected. If you want to collect data about this activity, you can collect it from the coordinator member or from all members. If you want to collect activity details such as the statement and the compilation environment information, you can do so by specifying the WITH DETAILS keyword. You can also use the AND VALUES keyword to have input data values (for those activities that have them) sent to the activities event monitor.

For example, assume that you have a work action set that is applied to a service superclass. You want to have activity data for all activities that are assigned to this work action written to the applicable event monitor, including all aggregate activity information, information about the compilation environment, and any input data values. You would specify:

```
COLLECT ACTIVITY DATA ON ALL WITH DETAILS AND VALUES
```

- If the work action set is associated with a service superclass or a workload, you can specify the COLLECT AGGREGATE ACTIVITY DATA keyword to collect aggregate activity data for activities that fall under the work class. If collected, aggregate activity data is captured and sent to the applicable event monitor. This information is collected periodically on an interval that is specified by the `wlm_collect_int` database configuration parameter.

For example, assume that you have a work action set that is applied to a service superclass. You want to have aggregate activity data for all activities that are assigned to this work action written to the applicable event monitor, including the base data, the activity data manipulation language (DML) estimated cost histogram, and the activity DML inter-arrival time histogram. You would specify

```
COLLECT AGGREGATE ACTIVITY DATA EXTENDED
```

- The histogram templates used by a COLLECT AGGREGATE ACTIVITY DATA work action to describe the histograms created for the corresponding work class. Specifying the histogram templates used by a work action adds the corresponding rows in the SYSCAT.HISTOGRAMTEMPLATEUSE, view which displays the histogram templates referenced by the service class or work action. For example, if you want to collect interarrival statistics for the default interarrival histogram template, you would specify:

```
INTERARRIVALTIME HISTOGRAM TEMPLATE SYSDEFAULTHISTOGRAM
```

For more information about histograms and histogram templates, see “Histograms in workload management” on page 258.
• Whether the work action is enabled or disabled. By default a work action is created as enabled, but you can specify whether it is enabled or disabled by using the ENABLE or DISABLE keyword. If the work action is disabled, the data server does not consider this work action when activities enter the database or service superclass (depending on the object you created the work action set for).

2. Commit your changes. When you commit your changes, the work action is added to the SYSCAT.WORKACTIONS view. If the work action is a threshold, the threshold is added to the SYSCAT.THRESHOLDS view.

A new work action takes effect in the database only after it is committed, and does not affect any database activities currently running.

**Altering a work action**

If you need to alter a work action, use the ALTER WORK ACTION SET statement.

**Before you begin**

To alter a work action, you require SQLADM, WLMADM or DBADM authority. To specify any clause other than a COLLECT clause, the authorization ID must include WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for additional prerequisites.

**Procedure**

To alter a work action:

1. Use the ALTER keyword of the ALTER WORK ACTION SET statement to change one or more of the following characteristics of the work action.

   • You can alter the work class to which the work action is applied. The work class must already exist in the work class set to which the work action set is applied.

   • If the work action maps to a service subclass, you can alter which service subclass the database activity is to be mapped. You can only change the mapping to a service subclass in the same service superclass. You cannot map to the default service subclass. You can also change whether nested activities in the activity are mapped to the same service subclass or not. For example, if the work action is currently defined as WITH NESTED, you can change this to WITHOUT NESTED. This change would cause the nested activities to be handled according to their activity type instead of automatically being mapped to the same service subclass as the parent activity. For example, if a CALL statement is mapped to service subclass SUBSC1, and the routine has an open cursor inside it, the open cursor might be mapped to a different service subclass if it falls under another work class that has another mapping work action applied to it.

   • You can alter the action type specified for the work action (that is, mapping, threshold, prevent execution, count activity, collect actions), but you must alter it to a valid work type. For example, if the work action is to map the activity to a service subclass, you cannot change the work action to a threshold, or the reverse. The reason is because, in this example, the work action set must have been applied to a service superclass in order to have a mapping action and threshold actions are not valid for work action sets applied to service superclasses. If you alter the type of a work action that is a threshold work action or alter the type of work action to a threshold, the following occurs:
If the work action was a threshold and has been changed to a non-threshold, the threshold is removed from the SYSCAT.THRESHOLDS view.

If the work action was not a threshold and has been changed to a threshold, a new threshold will be created in the SYSCAT.THRESHOLDS view.

**Note:** If the action is a threshold, you cannot alter the type of threshold to a different threshold. So, for example, if the work action was an SQLROWSRETURNED threshold, you cannot change it to a SQLTEMPSPACE threshold. In addition, you cannot change the work action type of an enabled CONCURRENTDBCOORDACTIVITIES work action threshold.

You can alter the histogram templates used by a COLLECT AGGREGATE ACTIVITY DATA work action to describe the histograms created for the corresponding work class. Updating the histogram templates used by a work action updates the corresponding rows in the SYSCAT.HISTOGRAMTEMPLATEUSE view, which displays the histogram templates referenced by the service class or work action. For more information about histograms and histogram templates, see “Histograms in workload management” on page 258.

You can alter whether you want to enable or disable the work action. By default, work actions are enabled. When enabled, the data server considers the work action for application against the activity that falls under the work class for the work action. If the work action is disabled, the data server ignores it.

2. Commit your changes. When you commit your changes, the work action is updated in the SYSCAT.WORKACTIONS view.

## Disabling a work action

You can disable a work action that you do not want applied to a work class. At runtime, the disabled work action is treated as if it does not exist.

**Before you begin**

To disable a work action, you require WLMADM or DBADM authority.

**Procedure**

To disable a work action:

1. Use one of the following statements, depending on whether you are creating or altering a work action set:
   - Use the DISABLE keyword and the ADD keyword of the CREATE WORK ACTION SET statement. For example:
     ```plaintext
     ADD WORK ACTION work-action-name ON WORK CLASS work-class-name ... DISABLE
     ```
   - Use the DISABLE keyword and the ALTER keyword of the ALTER WORK ACTION SET statement. For example:
     ```plaintext
     ALTER WORK ACTION work-action-name ... DISABLE
     ```

2. Commit your changes. When you commit your changes, the work action is updated in the SYSCAT.WORKACTIONS view.

## Dropping a work action

If you no longer require a work action, you can drop it from the work action set.
Before you begin

- To drop a work action, you require WLMADM or DBADM authority.
- See “Workload management DDL statements” on page 18 for additional prerequisites.

Procedure

To drop a work action:

1. Use the DROP keyword of the ALTER WORK ACTION SET statement. If you want to drop a CONCURRENTDBCOORDACTIVITIES threshold work action, you must disable the work action in one ALTER WORK ACTION SET operation, commit the change, verify that there are no queued activities, and then drop the threshold in a second ALTER WORK ACTION SET operation.
2. Commit your changes. When you commit your changes, the work action is removed from the SYSCAT.WORKACTIONS view. If the work action is a threshold work action, the threshold is also removed from the SYSCAT.THRESHOLDS view.

An altered work action set and work action only takes effect in the database after it is committed, and does not affect any database activities currently running.

Creating a work action set

To create a work action and a work action set, use the CREATE WORK ACTION SET statement.

Before you begin

To create a work action set, you require WLMADM or DBADM authority.

For additional prerequisites, see the following topics:

- “Workload management DDL statements” on page 18

When you create a work action set:

- You associate it with a work class set. The work class set must already exist.
- You also associate it with a database, a workload, or a service superclass. If you are associating the work action set with a service superclass, the service class must already exist. You cannot define the work action set for the default system service class (SYSDEFAULTSYSTEMCLASS), the default user class (SYSDEFAULTUSERCLASS) or the default maintenance service class (SYSDEFAULTMAINTENANCECLASS). If you are associating the work action set with a workload, the workload must exist in the current server. You cannot associate the work action set with the default administration workload (SYSDEFAULTADMWORKLOAD).

Procedure

To create a work action set:

1. Use the CREATE WORK ACTION SET statement with the following options:
   - Specify a name for the work action set. The name of the work action set must be unique in the database.
   - Specify the object with which the work action set is associated. You can specify a database, a workload, or a service superclass. If you specify that the work action set is associated with a database, none of the work actions in the
work action set can be mapping work actions or collect aggregate actions. If you specify that the work action set is associated with a service superclass, none of the work actions in the work action set can be thresholds. If you specify that the work action set is associated with a workload, then none of the work actions in the work action set can be mapping work actions. For example, to apply the work action set to the REPORTS service superclass, you would specify:

```
FOR SERVICE CLASS REPORTS
```

To apply the work action set to the database, you would specify:

```
FOR DATABASE
```

To apply the work action set to a workload named WL1, you would specify:

```
FOR WORKLOAD WL1
```

- Specify the work class set with which the work action set is associated. The work classes in the work class set classify the database activities that the work actions in the work action set will apply to. For example, to associate the work action set with the LARGEREADS work class set, you would specify:

```
USING WORK CLASS SET LARGEREADS
```

- Optional: Create one or more work actions for the work action set. For instructions, see “Creating a work action” on page 96.

- Optional: Specify whether the work action set is enabled or disabled. By default, the work action set is enabled. If the work action set is disabled, the data server does not consider this work action set (or any work actions in it) when activities are run.

2. Commit your changes. When you commit your changes the work action set is added to the SYSCAT.WORKACTIONSETS view.

A new work action set only takes effect in the database after it is committed, and does not affect any database activities currently running.

**Altering a work action set**

To add, alter, or drop a work action from a work action set, or to enable or disable the work action set, use the ALTER WORK ACTION SET statement.

**Before you begin**

To alter a work action set, you require SQLADM, WLMADM, or DBADM authority. To specify any clause other than a COLLECT clause, the authorization id must include WLMADM or DBADM authority.

For additional prerequisites, see the following topics:

- “Workload management DDL statements” on page 18
- Appendix A, “General naming rules,” on page 533

When you create a work action set to work with a specific work class set, you cannot change it to work with a different work class set because the work actions in the work action set have a dependency on the work classes in the work class set. If you want to change the work class set this work action set is to be applied to, you must drop and recreate the work action set.

You cannot change which object the work action set applies to because the type of work actions in the work action set depends on which object (database, workload, or service superclass) the work action set is defined for. If you want to change
which object the work action set is associated with, you must drop and recreate the work action set.

**Procedure**

To alter a work action set:

1. If you want to add a new work action to the work action set, use the ADD keyword. For information about the parameters that you can specify when adding a work action to a work action set, see “Creating a work action” on page 96.
2. If you want to alter an existing work action, use the ALTER keyword. For information about altering a work action, see “Altering a work action” on page 99.
3. If you want to drop a work action, use the DROP keyword. For information about dropping a work action from a work action set, see “Dropping a work action” on page 100.
4. You can enable a work action set that is not currently enabled, and the reverse. If you disable an enabled work action set, the data server ignores it after you commit your changes. For more information, see “Disabling a work action set.” If you enable the work action set, after you commit your changes, the work action set is applied to the next applicable activity that enters the database.

**Note:** Disabling a work action set does not disable the work actions within the work action set, but the work action set will no longer affect any work. At runtime, a disabled work action set is treated as if it does not exist. If you want to drop a work action set that contains a concurrency work action threshold, you must first disable the concurrency work action before the work action set can be dropped, because concurrency thresholds must be disabled before they can be dropped.

5. Commit your changes. When you commit your changes, the work action set is updated in the SYSCAT.WORKACTIONSETS view. The SYSCAT.WORKACTIONS view is updated for any added, altered, or dropped work actions.

**Disabling a work action set**

To disable a work action set, use the DISABLE keyword of the CREATE WORK ACTION SET statement or the ALTER WORK ACTION SET statement.

**Before you begin**

To disable a work action set, you require WLMADM or DBADM authority.

**About this task**

Disabling a work action set does not disable the work actions within the work action set, but the work action set will no longer affect any work. At runtime, a disabled work action set is treated as if it does not exist.

**Note:** If you want to drop a work action set that contains a concurrency work action threshold, you must first disable the concurrency work action before the work action set can be dropped, because concurrency thresholds must be disabled before they can be dropped.

For example, assume that you have a work action set called READACTIVITIES that is associated with a work class set called READCLASSES, and that work action set is defined for a service superclass called READSERVICECLASS. The SMALLREAD work action set has a work action in it that remaps all SELECT
If the READMECTIONS work action set is disabled, all SELECT statements are treated as though the READMECTIONS work action set does not exist, and are mapped to the default service subclass.

**Procedure**

To disable a work action set:
1. Use one of the following statements, depending on whether you are creating or altering a work action set:
   - To create a work action set that is disabled:
     ```sql
     CREATE WORK ACTION SET work-action-set-name ... DISABLE
     ```
   - To disable an already existing work action set:
     ```sql
     ALTER WORK ACTION SET work-action-set-name ... DISABLE
     ```
2. Commit your changes. When you commit your changes, the work action set is updated in the SYSCAT.WORKACTIONSETS view.

**Dropping a work action set**

Use the DROP WORK ACTION SET statement to drop a work action set.

**Before you begin**

To drop a work action set, you require WLMADM or DBADM authority.

**About this task**

Dropping a work action set drops the work action set and all work actions in it.

If the work action set contains a CONCURRENTDBCOORDACTIVITIES threshold work action, that work action must first be disabled before the work action set can be dropped.

**Procedure**

To drop a work action set:
1. Use the DROP WORK ACTION SET statement.
2. Commit your changes. When you commit your changes the work action set is removed from the SYSCAT.WORKACTIONSETS view. In addition, all work actions that were part of the work action set are removed from the SYSCAT.WORKACTIONS view. If the work action set contains threshold work actions, the thresholds are removed from the SYSCAT.THRESHOLDS view.

**Work actions and the work action set domain**

You can define a work action set for a database, a service superclass, or workload. The type of work actions that can be defined for a work action set depends on the type of object the work action set is defined for.

If the work action set is defined for a database, the work actions in the work action set must be any of the following actions:
- A threshold
  - The following thresholds apply to each individual activity in the matching work class:
    - ACTIVITYTOTALTIME
The following threshold applies to all activities in the matching work class as a group:
- CONCURRENTDBCOORDACTIVITIES
  The actual threshold is specified by the WHEN threshold-type clause. Multiple
  threshold work actions can be applied to a single work class if all the thresholds
  are of different types. If this action is specified, the threshold is applied to all
  database activities associated with the work class.

• PREVENT EXECUTION
  If this action is specified, all database activities that match the associated work
  class are not permitted to run.

• COLLECT ACTIVITY DATA
  If this action is specified, information about the database activities corresponding
  to the work class for which this work action is defined are written to the active
  ACTIVITIES event monitor when the activities complete execution. For more
  information, see “Collecting data for individual activities”.

• COUNT ACTIVITY
  If this action is specified, all database activity that maps to the associated work
  class causes the turnstile counter for that work class type to be incremented.
  (The turnstile counter for the work class is incremented by 1 each time an
  activity is associated with that work class). The COUNT ACTIVITY work action
  provides an efficient way to ensure this counter is updated. If no work action is
  applied to an activity corresponding to a work class, the work class activity
  counter is not incremented. Sometimes the only action you care about is
  obtaining a count of activities of a given type. For more information, see
  “Collecting data for individual activities”.

If the work actions in the work action set defined for a database are not any of
these actions, SQL4720N is returned.

If you are defining a work action set for a service superclass, the work actions in
the work action set must be any of the following actions:
• A mapping action
  You can map an activity to any service subclass in the service superclass except
  for the default service subclass. You specify the service subclass to map the
  activity to using the MAP ACTIVITY TO SERVICE CLASS keyword. Only one
  map work action in the work action set can be applied to the same work class.
• PREVENT EXECUTION
  Behavior is the same as for the database work action.
• COLLECT ACTIVITY DATA
  Behavior is the same as for the database work action.
• COLLECT AGGREGATE ACTIVITY DATA
  If this action is specified, aggregate database activity data that corresponds to
  the work class for which this work action is defined is collected.
• COUNT ACTIVITY
  Behavior is the same as for the database work action.
If the work actions in the work action set defined for a service superclass are not any of these actions, SQL4720N is returned.

If the work action set is defined for a workload, the work actions in the work action set must be any of the following actions:

- **A threshold**
  The following thresholds apply to each individual activity in the matching work class:
  - ACTIVITYTOTALTIME
  - CPUTIME
  - ESTIMATEDSQLCOST
  - SQLROWSREAD
  - SQLROWSRETURNED
  - SQLTEMPSPACE
  The following threshold applies to all activities in the matching work class as a group:
  - CONCURRENTDBCOORDACTIVITIES
  The actual threshold is specified by the `WHEN threshold-type` clause. Multiple threshold work actions can be applied to a single work class if all the thresholds are of different types. If this action is specified, the threshold is applied to all database activities associated with the work class.
- **PREVENT EXECUTION**
  Behavior is the same as for the database work action.
- **COLLECT ACTIVITY DATA**
  Behavior is the same as for the database work action.
- **COLLECT AGGREGATE ACTIVITY DATA**
  Behavior is the same as for the service superclass work action.
- **COUNT ACTIVITY**
  Behavior is the same as for the database work action.

If the work actions in the work action set defined for a workload are not any of these actions, SQL4720N is returned.

**Example: Applying work class activities**

The following figure shows an example of how the work classes in a work class set called LARGE ACTIVITIES are to be applied to both the database and a service superclass. To meet this objective, two work action sets, "Database large activities" and "Service class large activities" are created.

Although this example does not show it, you can also apply the classes in the LARGE ACTIVITIES work class set to a workload, by creating a work action set associated with the workload and then associating the work action set with the LARGE ACTIVITIES work class set.
The work action sets are as follows:

- **Database large activities** contains:
  - Work action: Concurrency threshold for large read
    - concurrency = 2, queued = 5
  - Work action: Rows returned threshold for large read
    - Rows returned > 1000
  - Work action: Count activity

- **Work class set: Large activities** contains:
  - Work class: Large reads
    - SELECT statements > 1000000
  - Work class: Large writes
    - UPDATE/INSERT/DELETE > 20000
  - Work class: LOAD

- **Work action set: Service class large activities** contains:
  - Work action: Map for large reads
    - Map large reads to SSC1
  - Work action: Prevent execution for large writes
  - Work action: Map for LOAD
    - Map LOAD to SSC2

- **Legend**
  - --- Associated with
  - Map to

Figure 17. Example of work actions, work actions sets, work classes, and work class set

The work action sets are as follows:

- Database large activities contains:
- Concurrency threshold for large reads, which permits two large reads to run concurrently, and five large reads to be queued
- Rows returned threshold for large reads, which prevents large reads from returning more than 1000 rows
- Count activity for load, which counts the number of times the load utility runs on the database.

• Service class large activities contains:
  - Map for large reads, which maps large reads to service subclass 1
  - Map for large writes, which prevents large writes from executing.
  - Map for LOAD, which maps loads to service subclass 2

A work action set does not have to contain an action for every work class in the work class set to which the work action set is applied. In addition, a work class can have more than one work action applied to it as long as the action types are different. A work class can have more than one threshold work action applied to it as long as the threshold types are different.

Thresholds that can be used in work actions

Work action sets that you define for databases can contain work actions that specify thresholds.

The following thresholds are supported:

• Aggregate threshold:
  - CONCURRENTDBCOORDACTIVITIES
  - CONCURRENTWORKLOADACTIVITIES

• Activity thresholds:
  - SQLTEMPSPACE
  - SQLROWSRETURNED
  - ACTIVITYTOTALTIME
  - ESTIMATEDSQLCOST
  - CPUPTIME
  - SQLROWSREAD

Application of work actions to database activities

One, and only one work action set can be applied to a database, a service superclass, or a workload.

When work is submitted to the data server, it is associated with a workload, either a user-defined workload or the default workload, then mapped to a service class.

The following figure shows the process of how a work action is applied to an activity.
A work action is assigned to an activity as follows:

1. When an activity is mapped to a service superclass or a service subclass, the data server checks whether an enabled database-level work action set exists.

2. If an enabled database-level work action set exists, the data server then checks whether the activity falls under any of the work classes in the work class set that the database-level work action set is associated with.

3. If the activity falls under a work class and that work class has any work actions applied to it, those work actions are applied to the activity.

4. The data server checks whether an enabled workload-level work action set exists. If it does, the data server then checks whether the activity falls under any of the work classes in the work class set that the workload-level work action set is associated with.

5. If the activity falls under a work class and that work class has any work actions applied to it, those work actions are applied to the activity.

6. Next, if the activity is mapped by the workload to a service superclass, the data server checks whether a work action set is applied to the service superclass.

7. If a work action set is applied to the service superclass, the data server then checks whether the activity falls under any of the work classes in the work class set that the service superclass-level work action set is associated with.
8. If the activity falls under a work class and that work class has any work actions applied to it, those work actions are applied to the activity.

Note that, if a mapping work action is applied to a stored procedure, then depending on whether the WITH NESTED or WITHOUT NESTED clause is specified in the work action definition, child activities of a stored procedure can run in the same service subclass or in different service subclasses than the parent activity.

In the following situations an activity is not affected by a work action set:

- Activities fall in the default system (SYSDEFAULTSYSTEMCLASS) and default maintenance (SYSDEFAULTMAINTENANCECLASS) service classes.
- Activities are assigned to the default administration workload, SYSDEFAULTADMWORKLOAD.
- Activities are inside a load operation. The load operation itself does go through work action set evaluation.
- Child activities of system stored procedures. The only exception is the SYSPROC.ADMIN_CMD stored procedure. Child activities of SYSPROC.ADMIN_CMD go through work action set evaluation.
- The work action set is disabled.
- The workload maps the activity directly to a service subclass.

**Concurrency control at the workload level using work action sets**

At the workload level, you control the concurrency of incoming work using concurrency thresholds applied with a work action set defined on the workload. You can combine workload-level concurrency control with priority aging at the service class level.

Figure 1 illustrates an example scenario using work action sets to control the concurrency of incoming work based on the source of the connection while all work in the database is controlled using priority aging.
In the example scenario, two workloads are created to identify and differentiate the work coming from different sources. Connections to the database from the sales, accounting, and IT departments are mapped to the Regular workload. Connections to the database from management and critical applications are mapped to the Important workload. Work from the Important workload has higher priority and needs to be able to complete within the shortest amount of time. To ensure the database has sufficient capacity for work in the Important workload, concurrency thresholds are placed on the work in the Regular workload. A workload level work action set, called Regular workload level work action set, is created on the Regular workload and is applied to a work class set that has two work classes. Load activities are mapped to one work class, while all other activities are mapped to the other work class. A CONCURRENTDBCOORDACTIVITIES threshold is created as a work action in the Regular workload level work action set to allow only one load activity in the system at a time while queuing the other load activities. In addition, another CONCURRENTDBCOORDACTIVITIES threshold is created as a work action in the Regular workload level work action set to allow a maximum of 500 concurrent activities, while activities exceeding the maximum are queued.

Figure 19. Concurrency control at the workload level using work action sets

In the example scenario, two workloads are created to identify and differentiate the work coming from different sources. Connections to the database from the sales, accounting, and IT departments are mapped to the Regular workload. Connections to the database from management and critical applications are mapped to the Important workload. Work from the Important workload has higher priority and needs to be able to complete within the shortest amount of time. To ensure the database has sufficient capacity for work in the Important workload, concurrency thresholds are placed on the work in the Regular workload. A workload level work action set, called Regular workload level work action set, is created on the Regular workload and is applied to a work class set that has two work classes. Load activities are mapped to one work class, while all other activities are mapped to the other work class. A CONCURRENTDBCOORDACTIVITIES threshold is created as a work action in the Regular workload level work action set to allow only one load activity in the system at a time while queuing the other load activities. In addition, another CONCURRENTDBCOORDACTIVITIES threshold is created as a work action in the Regular workload level work action set to allow a maximum of 500 concurrent activities, while activities exceeding the maximum are queued.
Note: If a threshold action of CONTINUE is specified for a queuing threshold, it effectively makes the size of the queue unbounded, regardless of any hard value specified for the queue size.

Connections to the database from both the Regular and Important workloads are mapped to the Priority Aging service superclass. This service superclass is created to implement priority aging that favors short activities. The Priority Aging service class work action set is created for the Priority Aging service superclass to separate the long-running load activities from all the short-running activities. All activities, other than load, are mapped to the Short service subclass. The Short service subclass is configured to have the highest agent, prefetch, and buffer pool priorities. A CPUTIMEINSC threshold is created on the Short service subclass to remap an activity to the Medium service subclass after it consumes more than 1 second of processor time in the Short service subclass. The Medium service subclass has medium agent, prefetch, and buffer pool priorities. A CPUTIMEINSC threshold is created on the Medium service subclass to remap an activity to the Long service subclass after it consumes more than 5 seconds of processor time in the Medium service subclass. The Long service subclass has the lowest agent, prefetch, and buffer pool priorities. Load activities are mapped directly to the Long service subclass by the Priority Aging service class work action set because load activities can be long running, resource intensive, and less time critical for completion.

Workload and work action set comparison

Depending on the type of control that you want to maintain over your database activities, you can use workloads by themselves or both workloads and work classes (when used with work actions) to map activities to service classes.

With workloads, requests are identified and assigned to a service class based on connection attributes. Workloads are the primary method for routing work to a specific DB2 service class for execution. If you want to further refine how requests are identified, you can use work classes to classify the activities based on their type and other activity attributes. For example, you can classify READ activities, WRITE activities, and LOAD activities into different work classes and have each activity type treated differently.

If you use work classes (which are grouped into work class sets), you can use work actions to exercise control over the different types of activities. For example, you can use a work action in a work action set defined on a service superclass to map a specific type of activity to a service subclass. In a work action set defined on the database or workload, you can define a work action to apply a control known as a threshold to ensure that the same type of activity does not exceed certain conditions.

Work actions are grouped into work action sets. A single work action set can apply to activities in the database, to activities in a service superclass, or to activities in a workload. However, the same work action set cannot apply to more than one object. Work class sets and work action sets work together. That is, a work class must exist for categorizing an activity as a specific type of work before a work action can be applied to it. A work class set can be associated with more than one work action set, but a work action set can be associated with only one work class set.

Figure 1 shows an example of a DB2 workload manager implementation that uses workloads and work action sets. In this figure, assume that a request is assigned to
workload WL_A based on the connection attributes of the connection that submitted the request. Workload WL_A specifies that the request is to be executed in service superclass SC_A. Assume that a work class in work class set WCS_1 matches the type of work that the request that is associated with workload WL_A is going to perform.

Now assume that an activity that does not update the catalogs (a READ activity) enters the system. The database-level work action set WAS_1 (that is associated with work class set WCS_1) contains a work action that is applied to the READ work class. The request is then mapped to service superclass SC_A (by workload WL_A). Here, the request encounters the service superclass-level work action set WAS_2, which is also associated with work class set WCS_1, and applies to activities in service superclass SC_A. This work action set contains a mapping work action, which is also applied to the READ work class so that all READ activities will be mapped to service subclass SSC_1a in service superclass SC_A.

A somewhat similar situation occurs with the request that is associated (again, based on its connection attributes) with workload WL_B. Workload WL_B maps activities to service superclass SC_B. Assume that the request is for a LOAD activity and that work class set WCS_2 contains a work class that applies to LOAD activities. Work class set WCS_2 is associated with the service superclass-level work action set WAS_3, which applies to activities in service superclass SC_B. Assume that work action set WAS_3 contains a mapping work action that is applied to the LOAD work class, so that when the LOAD activity is mapped to service superclass SC_B by workload WL_B, it will then be mapped by the work action to service subclass SSC_1b for execution.

The purpose of workload WL_C in this example is to map incoming requests directly to service subclass SSC_1b, independent of the service superclass-level work action set WAS_3 and its mapping work action. If an incoming request is associated with workload WL_C that is a LOAD activity, then this request is also mapped directly to service subclass SSC_1b for execution, and is unaffected by the mapping work action that applies to the LOAD work class.
Example: Using a database work action set and database threshold

This example shows different approaches to using work action sets and thresholds to control the resources consumed by DB2 activities. Before creating DB2 workload management objects, you need to understand how they are used.

Assume that you have a work class set called ALLSQL, and it contains the following work classes in this order:

1. SMALLDML, which is for all DML-type SQL statement that have an estimated cost of less than 1 000 timerons

Figure 20. Workloads and work action sets
2. MEDDML, which is for all DML-type SQL statements that have an estimated cost between 1,000 and 20,000 timerons
3. LARGEDML, which is for all DML-type SQL statements that have an estimated cost greater than 20,000 timerons
4. ALLDDL, which is for all DDL-type SQL statements
5. ALLACTIVITY, which is for all database activity

The following SQL statements create the work class set and the work classes:

```sql
CREATE WORK CLASS SET ALLSQL
(WORK CLASS SMALLDML WORK TYPE DML FOR TIMERONCOST FROM 0 TO 1000,
 WORK CLASS MEDDML WORK TYPE DML FOR TIMERONCOST FROM 1001 TO 20000,
 WORK CLASS LARGEDML WORK TYPE DML FOR TIMERONCOST FROM 20001 TO UNBOUNDED,
 WORK CLASS ALLDDL WORK TYPE DDL,
 WORK CLASS ALLACTIVITY WORK TYPE ALL)
```

These work classes already have work actions, such as COUNT ACTIVITY, COLLECT, and thresholds (that are not ACTIVITYTOTALTIME thresholds) applied to them.

Assume that you want to permit large DML activities to run for no longer than 5 hours. All other SQL can take no longer than 30 minutes to run. The following two examples show possible methods for accomplishing this objective.

**Method 1**

One method is to set up a work action set at the database level containing work actions that specify an ACTIVITYTOTALTIME threshold for each work class, as follows:

<table>
<thead>
<tr>
<th>Work action</th>
<th>Work class applied to</th>
<th>Threshold type and value</th>
<th>Actions</th>
</tr>
</thead>
</table>
| SMALLDMLTIMEALLOWED    | SMALLDML              | ACTIVITYTOTALTIME > 30 MINUTES | • Stop execution  
|                        |                       |                          | • Collect activity data          |
| MEDDMLTIMEALLOWED      | MEDDML                | ACTIVITYTOTALTIME > 30 MINUTES | • Stop execution  
|                        |                       |                          | • Collect activity data          |
| LARGEDMLTIMEALLOWED    | LARGEDML              | ACTIVITYTOTALTIME > 5 HOURS | • Stop execution  
|                        |                       |                          | • Collect activity data          |
| ALLDDLTIMEALLOWED      | ALLDDL                | ACTIVITYTOTALTIME > 30 minutes | • Stop execution  
|                        |                       |                          | • Collect activity data          |
| ALLACTIVITYTIMEALLOWED | ALLACTIVITY           | ACTIVITYTOTALTIME > 30 minutes | • Stop execution  
|                        |                       |                          | • Collect activity data          |

The SQL statements for this method are:

```sql
CREATE WORK ACTION SET WASNICK FOR DATABASE USING WORK CLASS SET WCSNICK
(WORK ACTION SMALLDMLTIMEALLOWED ON WORK CLASS SMALLDML
 WHEN ACTIVITYTOTALTIME > 30 MINUTES COLLECT ACTIVITY DATA STOP EXECUTION,
 WORK ACTION MEDDMLTIMEALLOWED ON WORK CLASS MEDDML
 WHEN ACTIVITYTOTALTIME > 30 MINUTES COLLECT ACTIVITY DATA STOP EXECUTION,
 WORK ACTION LARGEDMLTIMEALLOWED ON WORK CLASS LARGEDML
 WHEN ACTIVITYTOTALTIME > 5 HOURS COLLECT ACTIVITY DATA STOP EXECUTION,
 WORK ACTION ALLDDLTIMEALLOWED ON WORK CLASS ALLDDL
```
Method 2

Another method might be to use only one work class, LARGEDML, then create a work action set for the database that has one work action, LARGEDMLTIMEALLOWED, applied to the work class.

Table 33. LARGEDMLTIMEALLOWED work action applied to the LARGEDML work class

<table>
<thead>
<tr>
<th>Work action</th>
<th>Work class applied to</th>
<th>Threshold type and value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>LARGEDMLTIMEALLOWED</td>
<td>LARGEDML</td>
<td>ACTIVITYTOTALTIME &lt; 5 HOURS</td>
<td>• Stop execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Collect activity data</td>
</tr>
</tbody>
</table>

You would then apply an ACTIVITYTOTALTIME threshold of less than 31 MINUTES to the database. Using this method, only those activities that correspond to the LARGEDML work class have the 5 hour threshold applied to them. Other activities will have the ACTIVITYTOTALTIME database time threshold of less than 31 minutes applied to them.

The SQL statements for this method are:

CREATE WORK ACTION SET WASNICK FOR DATABASE USING WORK CLASS SET WCSNICK
(WORK ACTION LARGEDMLTIMEALLOWED ON WORK CLASS LARGEDML
WHEN ACTIVITYTOTALTIME > 5 HOURS COLLECT ACTIVITY DATA STOP EXECUTION)

CREATE THRESHOLD THTEST FOR DATABASE ACTIVITIES ENFORCEMENT DATABASE
WHEN ACTIVITYTOTAL TIME > 30 MINUTES COLLECT ACTIVITY DATA STOP EXECUTION

Example: Using work action sets to determine the types of work being run

Using work class sets, work classes, work action sets, work actions, and some of the DB2 workload manager monitoring features, you can determine the different types of work running on your system, and the distribution of the work.

One way of accomplishing this task is described here. First create a work class set that contains work classes for the different types of work you are interested in. For example, if you want to know how many READ activities, WRITE activities, DDL activities, and LOAD activities are running on your system, you would create a work class set, ACTIVITYTYPES, as in the following example:

CREATE WORK CLASS SET ACTIVITYTYPES
(WORK CLASS READWC WORK TYPE READ,
WORK CLASS WRITENC WORK TYPE WRITE,
WORK CLASS DDLWC WORK TYPE DDL,
WORK CLASS LOADWC WORK TYPE LOAD)

Next, you would create a database-level work action set, COUNTACTIONS, to apply to the ACTIVITYTYPES work class set. The work action set would contain a COUNT ACTIVITY work action for each work class in the ACTIVITYTYPES work class set, as in the following example:

CREATE WORK ACTION SET COUNTACTIONS FOR DATABASE USING WORK CLASS SET ACTIVITYTYPES
(WORK ACTION COUNTREAD ON WORK CLASS READWC COUNT ACTIVITY,
WORK ACTION COUNTWRITE ON WORK CLASS WRITENC COUNT ACTIVITY,
WORK ACTION COUNTDDL ON WORK CLASS DDLWC COUNT ACTIVITY,
WORK ACTION COUNTLOAD ON WORK CLASS LOADWC COUNT ACTIVITY)
After a sufficient amount of time has passed, you can determine the number of each type of activity that has run by using the WLM_GET_WORK_ACTION_SET_STATS table function:

```
SELECT SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
      LAST_RESET,
      SUBSTR(WORK_CLASS_NAME,1,15) AS WORK_CLASS_NAME,
      SUBSTR(CHAR(ACT_TOTAL),1,14) AS TOTAL_ACTS
FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS(CAST(NULL AS VARCHAR(128)), -2))
      AS WASSTATS WHERE WORK_ACTION_SET_NAME = 'COUNTACTIONS'
ORDER BY WORK_CLASS_NAME, MEMB
```

Control of work with thresholds

You can use thresholds to maintain stability in the system by identifying work that behaves abnormally. You can identify abnormal behavior predictively, before the work begins running, based on the projected impact. You can also identify abnormal behavior reactively, as the work is running and consuming resources.

An example of work that can be controlled with thresholds is a query that consumes large amounts of processor time at the expense of all other work running on the system. Such a query can be controlled either before it even begins executing, based on estimated cost, or after it has begun executing and is consuming more than the permitted amount of resources.

Types of thresholds

Connection thresholds

If you want to limit how long a connection can sit idle, use a connection threshold. These thresholds can be used to detect connections that sit idle for too long.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTIONIDLETIME</td>
<td>Controls the amount of time that a connection sits idle and is not working on behalf of user requests. Use this threshold to detect inefficient use of data server resources and application wait conditions.</td>
</tr>
</tbody>
</table>

Unit of work thresholds

If you want to limit how long a unit of work executes, use a unit of work threshold. These thresholds limit the maximum amount of time that a unit of work may spend in the DB2 engine, and can be used to detect units of work that take too long to complete.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UOWTOTALTIME</td>
<td>Controls the amount of time that a unit of work takes to execute.</td>
</tr>
</tbody>
</table>

Activity thresholds

If you want to limit the impact that specific activities can have on how the data server is running, activity thresholds provide you with one of the means you can use. Excess execution time, abnormally high volumes of data returned, or abnormally high amounts of resources consumed are all examples of warning flags that potentially troublesome activities could be consuming excessive resources, which you can control with activity thresholds.
Table 36. Activity thresholds

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVITYTOTALTIME</td>
<td>Controls the amount of time that any given activity can spend from submission to completion, for both execution and queue time. Use this threshold to detect jobs that are taking an abnormally long time to complete.</td>
</tr>
<tr>
<td>CPUTIME</td>
<td>Controls the maximum amount of combined user and system processor time that an activity may consume on a particular member during the execution of the activity. Use this threshold to detect and control activities which are consuming excessive processor resources.</td>
</tr>
<tr>
<td>CPUTIMEINSC</td>
<td>Controls the maximum amount of combined user and system processor time that an activity may consume on a particular member while executing in a specific service subclass. Use this threshold to detect and control activities which are consuming excessive processor resources within the current service class.</td>
</tr>
<tr>
<td>DATATAGINSC</td>
<td>Controls the data that can be touched or not touched by an activity while running a particular service subclass.</td>
</tr>
<tr>
<td>ESTIMATEDSQLCOST</td>
<td>Controls DML activities that the query optimizer determines to have a large estimated cost. Use this threshold to predict potentially resource-heavy SQL before it starts executing on the system and identifying poorly written SQL.</td>
</tr>
<tr>
<td>SQLROWSREAD</td>
<td>Controls the maximum number of rows which can be read on any member by an activity. Use this threshold to detect and control activities which are reading an excessive number of rows.</td>
</tr>
<tr>
<td>SQLROWSREADINSC</td>
<td>Controls the maximum number of rows which can be read by an activity on a particular member while executing in a specific service subclass. Use this threshold to detect and control activities which are reading an excessive number of rows within the current service class.</td>
</tr>
<tr>
<td>SQLROWSRETURNED</td>
<td>Controls the number of rows returned when executing SQL. Use this threshold to identify when the amount of data exceeds a reasonable volume.</td>
</tr>
<tr>
<td>SQLTEMPSPACE</td>
<td>Controls the amount of temporary table space a given activity can consume on a member. Use this threshold to prevent certain SQL statements from using up a disproportionate amount of temporary space, impeding the progress of other work.</td>
</tr>
</tbody>
</table>

The data server considers requests from utilities such as REORGCHK, IMPORT, and EXPORT to be user logic, and thus these requests are subject to any defined thresholds.

**Aggregate thresholds**

If you want to control the collective impact that certain activities, workloads, or connections can have on your data server, you can define aggregate thresholds. Aggregate thresholds often, but not always, enforce concurrency control in cases where you need to limit the number of certain activities running at the same time. Some aggregate thresholds have a built-in queue and are known as queuing thresholds.

Table 37. Aggregate thresholds

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGGSQLEMPSPACE</td>
<td>Controls the maximum amount of system temporary table space that can be consumed in total across all activities in the service subclass. Use this threshold to detect and control activities that belong to a service subclass whose activities are consuming too much system temporary table space in aggregate across all of the activities in the service subclass.</td>
</tr>
<tr>
<td>CONCURRENTWORKLOADOCCURRENCES</td>
<td>Controls the number of active occurrences of a workload that can run on a coordinator member at the same time. Use to control the spread of connections from a specific source.</td>
</tr>
</tbody>
</table>
### Table 37. Aggregate thresholds (continued)

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCURRENTWORKLOADACTIVITIES</td>
<td>Controls the number of individual activities that can run within a workload occurrence. Use to limit work within an individual workload occurrence.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES</td>
<td>Controls the number of concurrent activities in the domain that the threshold is associated with (database, work action, service superclass, or service subclass).</td>
</tr>
<tr>
<td>TOTALMEMBERCONNECTIONS</td>
<td>Controls the number of database connections to a given member that can be established at the same time. Use to prevent a given member from becoming overloaded.</td>
</tr>
<tr>
<td>TOTALSCMEMBERCONNECTIONS</td>
<td>Controls the number of database connections to a given member for work executing within a given service class at the same time. Similar to the total member connections but more granular because the connection is linked to a service class.</td>
</tr>
</tbody>
</table>

For those aggregate thresholds that support it, concurrency control is provided through a system of execution 'tickets.' Each incoming activity must claim a ticket from the applicable concurrency threshold before it can begin executing. Once all tickets are consumed, additional activities are queued until a ticket becomes available or an error is returned, depending on how you defined the threshold. If the concurrency threshold has queuing enabled, then a ticket passes from an activity that has finished execution to another activity that is in the queue. This activity can then leave the queue and begin execution. How many tickets are available per concurrency threshold depends on how you defined the threshold. For example, if you defined a CONCURRENTDBCOORDACTIVITIES threshold to limit the number of concurrently running database activities to 10, then there are 10 execution tickets available.

For stored procedures, activity thresholds are applied to the stored procedure and to its child activities. Concurrency thresholds are only applied to the child activities of the stored procedure and not to the CALL statement itself.

### Taking action when thresholds are violated

The action that is taken dynamically when a threshold is violated depends on how you define the threshold.

**Stop execution (STOP EXECUTION)**

A common action when a threshold is violated is to stop the activity from executing. In this case, an error code is returned to the submitting application indicating that the threshold was violated. Note that for TOTALMEMBERCONNECTIONS and TOTALSCMEMBERCONNECTIONS thresholds, a STOP EXECUTION action prevents a connection from being established. For CONNECTIONIDLETIME thresholds, the connection is closed. For CONCURRENTWORKLOADOCCURRENCES, a new workload occurrence is prevented from being created. For all activity-related thresholds, the activity is stopped from continuing to execute. If a THRESHOLDVIOLATIONS event monitor is active, a record is written to the event monitor indicating that the threshold was violated.

**Continue execution (CONTINUE)**

In some situations, stopping the execution of an activity is too harsh a response. A preferable response is to permit the activity to continue to run...
and to collect the relevant data for an administrator to perform future analysis to determine how to prevent this condition from happening again. In this situation, no error code is returned to the submitting application. If the action is to continue, the user receives no indication that the threshold was violated. If a THRESHOLDVIOLATIONS event monitor is active, a record is written to the event monitor. If a CONTINUE threshold action is specified for a queuing threshold, this effectively renders the size of the queue unbounded, regardless of any hard value you include.

**Force the application (FORCE APPLICATION)**

When the UOWTOTALTIME threshold is violated, you can specify that the local or remote users or applications are forced off the system. This benefits other applications competing for server resources.

**Remap the activity (REMAP ACTIVITY TO)**

When an activity violates a certain limit, you may simply wish to assign different resource controls to it but to let the activity continue executing otherwise. Such a response permits you to dynamically raise or lower the amount of resources an activity can consume throughout its lifetime. In this case, an already executing activity is permitted to continue with no indication to the user or application that the threshold was violated, although the activity now runs with different resources available to it. Remapping is available with any of the in-service-class thresholds like CPUTIMEINSC, SQLROWSREADINSC, and DATATAGINSC.

**Collect data (COLLECT ACTIVITY DATA)**

When some thresholds are violated, data is collected. By default the fact that an activity threshold was violated is recorded in an activated threshold violations event monitor. If you want more detailed information about the activity that violated the threshold, you can request that information for the activity be written to the active event monitor for activities when the activity completes execution using the COLLECT ACTIVITY DATA clause.

**Threshold domain and enforcement scope**

Each threshold operates on a domain. Only activities taking place in the domain of a threshold can be affected by that threshold.

The following threshold domains exist:

- Database
- Service superclass
- Service subclass
- Work action
- Workload
- Statement

Within each of these threshold domains, a threshold has a scope over which it is enforceable, such as a single workload occurrence, a member, the execution of a specific statement, or all members. This is known as the enforcement scope of the threshold. For example: Service class aggregate thresholds can have one of two enforcement scopes: database and member. An example of an aggregate threshold that applies only at the member level is the maximum number of concurrent connections for a service superclass on a member (TOTALSCMEMBERCONNECTIONS). Similarly, the following table shows that you can specify the processor time threshold (CPUTIME) at the database,
superclass, subclass, work action or workload domain and that it is enforced per member. That is, the upper boundary specifies the maximum amount of user and system processor time per member that an activity may use.

### Table 38. Threshold domains and enforcement scopes

<table>
<thead>
<tr>
<th>Threshold domain</th>
<th>Thresholds with database enforcement scope</th>
<th>Thresholds with member enforcement scope</th>
<th>Thresholds with workload occurrence enforcement scope</th>
</tr>
</thead>
</table>
| **Database**     | • “ACTIVITYTOTALTIME threshold” on page 129  
                  • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “CONCURRENTDBCOORDACTIVITIES threshold” on page 139  
                  • “CONNECTIONIDLETIME threshold” on page 128  
                  • “ESTIMATEDSQLCOST threshold” on page 135  
                  • “SQLROWSRETURNED threshold” on page 136  
                  • “UOWTOTALTIME threshold” on page 145 | • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “CPUTIME threshold” on page 130  
                  • “SQLROWSREAD threshold” on page 134  
                  • “SQLTEMPSPACE threshold” on page 137 | Not applicable |
| **Work action**  | • “ACTIVITYTOTALTIME threshold” on page 129  
                  • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “ESTIMATEDSQLCOST threshold” on page 135  
                  • “SQLROWSRETURNED threshold” on page 136 | • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “CPUTIME threshold” on page 130  
                  • “SQLROWSREAD threshold” on page 134  
                  • “SQLTEMPSPACE threshold” on page 137 | Not applicable |
| **Service superclass** | • “ACTIVITYTOTALTIME threshold” on page 129  
                  • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “CONNECTIONIDLETIME threshold” on page 128  
                  • “ESTIMATEDSQLCOST threshold” on page 135  
                  • “SQLROWSRETURNED threshold” on page 136 | • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “CPUTIME threshold” on page 130  
                  • “SQLROWSREAD threshold” on page 134  
                  • “SQLTEMPSPACE threshold” on page 137 | Not applicable |
| **Service subclass** | • “ACTIVITYTOTALTIME threshold” on page 129  
                  • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “ESTIMATEDSQLCOST threshold” on page 135  
                  • “SQLROWSRETURNED threshold” on page 136 | • “CONCURRENTDBCOORDACTIVITIES threshold” on page 136  
                  • “CPUTIME threshold” on page 130  
                  • “SQLROWSREAD threshold” on page 134  
                  • “SQLTEMPSPACE threshold” on page 137 | Not applicable |
| **Workload**     | • “ACTIVITYTOTALTIME threshold” on page 129  
                  • “ESTIMATEDSQLCOST threshold” on page 135  
                  • “SQLROWSRETURNED threshold” on page 136  
                  • “UOWTOTALTIME threshold” on page 145 | • “CONCURRENTWORKLOADOCCURRENCES threshold” on page 142  
                  • “CPUTIME threshold” on page 130  
                  • “SQLROWSREAD threshold” on page 134  
                  • “SQLTEMPSPACE threshold” on page 137 | • “CONCURRENTWORKLOADACTIVITIES threshold” on page 140 |
| **Statement**    | • “ACTIVITYTOTALTIME threshold” on page 129  
                  • “ESTIMATEDSQLCOST threshold” on page 135  
                  • “SQLROWSRETURNED threshold” on page 136  
                  • “UOWTOTALTIME threshold” on page 145 | • “CPUTIME threshold” on page 130  
                  • “SQLROWSREAD threshold” on page 134  
                  • “SQLTEMPSPACE threshold” on page 137 | Not applicable |

* Database enforcement scope is only allowed in environments other than DB2 pureScale® environments. In a DB2 pureScale environment, only member enforcement scope is allowed.

### Threshold evaluation order

Thresholds are evaluated in a specific order when you define them on a database.

The following thresholds are evaluated before all others:

- **TOTALMEMBERCONNECTIONS**. This threshold is evaluated when a new connection is made to a database.
- **CONCURRENTWORKLOADOCCURRENCES**. This threshold is evaluated when a new workload occurrence is started for a workload definition that has this threshold applied to it.
- **TOTALSCMEMBERCONNECTIONS**. This threshold is evaluated when a connection joins a service class (either a new connection or a transfer between service classes as a result of workload reassignment).

All other thresholds are based on recognized activities resulting from an SQL statement or the execution of a utility as the load utility and are evaluated in the following order:
Predictive thresholds

Predictive thresholds are checked before reactive thresholds, because they affect whether a database activity can start to run.

The sequence in which predictive thresholds are evaluated is as follows. If you do not define a particular threshold, its step is skipped. Also, the steps described might be combined at run time for performance reasons.

1. Check if a CONCURRENTWORKLOADACTIVITIES threshold exists and if so, whether it has been violated. If the threshold is violated, the corresponding action is taken. If applicable, move to the next step.
2. Check if an ESTIMATEDSQLCOST threshold exists and if so, whether it has been violated. If you define this threshold in more than one domain, the threshold is resolved according to the scope resolution rules (see “Domain precedence for activity thresholds” on page 129 for more information). The result of this operation is one value of ESTIMATEDSQLCOST applicable to the activity. If the threshold is violated, the corresponding action is taken.
3. Check if a CONCURRENTDBCOORDACTIVITIES threshold exists for the workload work action set threshold domain and if so, whether it has been violated. If the threshold is violated, the corresponding action is taken.
4. Check if a CONCURRENTDBCOORDACTIVITIES threshold exists for the database work action set threshold domain and if so, whether it has been violated. If the threshold is violated, the corresponding action is taken.
5. Check if a CONCURRENTDBCOORDACTIVITIES threshold exists for the service subclass threshold domain and if so, whether it has been violated. If the threshold is violated, the corresponding action is taken.
6. Check if a CONCURRENTDBCOORDACTIVITIES threshold exists for the service superclass threshold domain and if so, whether it has been violated. If the threshold is violated, the corresponding action is taken.
7. Check if a CONCURRENTDBCOORDACTIVITIES threshold exists for the database threshold domain and if so, whether it has been violated. If the threshold is violated, the corresponding action is taken.

Concurrency threshold considerations: The evaluation order for concurrency thresholds does not follow the hierarchy used for resolving activity thresholds. An activity must pass through each defined concurrency threshold before it is permitted to execute.

For concurrency thresholds, thresholds for workload-level work action sets are checked first and database-level work action sets are checked second. Thresholds for work action sets are checked first, in order to avoid work action set thresholds on particular types of work blocking work of other types, which would affect concurrency. For example, by checking database-level work action set concurrency thresholds first, the following situation is avoided.

Assume that the following thresholds are defined:
- A database work action set containing a work action concurrency threshold for LOAD activities is defined with a value of 1.
- The service superclass S1 concurrency limit is set to 10.
Also, assume that one LOAD activity is already running in the database (under any service superclass) and nine activities are already running in service superclass S1. A second new LOAD activity enters as the 10th activity. If the activity threshold scope resolution hierarchy were used during threshold evaluation, the incoming LOAD activity would not violate the service class threshold, increasing the concurrency to 10. The LOAD activity is then evaluated against the database-level work action threshold concurrency limit, which is violated because a LOAD activity is already running in the database and the work action threshold concurrency value is only 1. The second LOAD activity is then queued.

Any new activity arriving into service superclass S1 is now queued (because the service class concurrency limit is already reached). The work action threshold queue is affecting the service class, which is undesirable because activities trying to run in the service class do not necessarily have a relationship with the work action threshold condition (for example, an insert operation trying to run in service superclass S1 should not have to wait on a LOAD activity that is queued because of a database-level work action threshold condition). Therefore, to avoid this type of situation, the database-level work action concurrency threshold is checked before service subclass, service superclass, and database concurrency thresholds. Because the database-level work action set concurrency threshold is checked first, the 10th activity in the service class (which happens to be a LOAD activity) is blocked at the database work action threshold level before it can attempt to consume one spot in the service superclass S1.

**Reactive thresholds**

Reactive thresholds are evaluated in a discrete fashion when an activity is executing and no specific order is used to evaluate reactive thresholds. The following reactive thresholds are available to you:

- ACTIVITYTOTALTIME
- AGGSQLEMPSPACE
- CONNECTIONIDLETIME
- CPUTIME
- CPUTIMEINSC
- DATATAGINSC
- SQLTEMPSPACE
- SQLROWSREAD
- SQLROWSREADINSC
- SQLROWSRETURNED
- UOWTUTALTIME

**Creating a threshold**

Create thresholds using the DDL statement CREATE THRESHOLD (or the CREATE WORK ACTION SET statement). You create a threshold to impose a limit on resource consumption.

**Before you begin**

To create a threshold, you require WLMADM or DBADM authority.

See the following topics for more information about prerequisites:

- “Workload management DDL statements” on page 18
Naming rules

To create a threshold for a work action set, use the CREATE WORK ACTION SET statement or the ALTER WORK ACTION SET statement with the ADD WORK ACTION keywords. For more information, see CREATE WORK ACTION SET statement or ALTER WORK ACTION SET statement.

Procedure

To create a threshold:
1. Issue the CREATE THRESHOLD statement, specifying one or more of the following properties for the threshold:
   - The name of the threshold.
   - The threshold domain. The threshold domain is the database object that the threshold is both attached to and operates on. The domain that applies depends on the type of threshold, see “Threshold domain and enforcement scope” on page 120 for more information.
   - The enforcement scope for the threshold. The threshold scope is the enforcement range of the threshold in its domain. The enforcement scope that applies depends on the type of threshold, see “Threshold domain and enforcement scope” on page 120 for more information.
   - Optional: Disable the threshold when it is created. By default a threshold is created as enabled. If you create the threshold as disabled and want to enable it later, use the ALTER THRESHOLD statement.
   - The threshold predicate to specify the type of threshold and the maximum value permitted. When the maximum value is violated, the action specified for the threshold is enforced. For more information on which thresholds are available to you, see “Connection thresholds” on page 128, “Activity thresholds” on page 128, “Aggregate thresholds” on page 137, and “Unit of work thresholds” on page 144.
   - The actions to be taken if the maximum value for the threshold is exceeded. The actions consist of a mandatory action that affects the execution of the activity (STOP EXECUTION, CONTINUE, FORCE APPLICATION, or REMAP ACTIVITY TO) and an optional collect activity action (COLLECT ACTIVITY DATA). The options you specify for the collect activity action determine what information is collected for the activity that caused the threshold boundary to be violated.
2. Commit your changes. When you commit your changes, the threshold is added to the SYSCAT.THRESHOLDS view.

Altering a threshold

Alter thresholds using the ALTER THRESHOLD statement. You might alter a threshold to modify the limit imposed on a specific resource.

Before you begin

To alter a threshold, you require SQLADM, WLMADM, or DBADM authority. To specify any clause other than a COLLECT clause, the authorization ID must include WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.
To alter a threshold for a work action set, use the ALTER WORK ACTION SET statement with the ADD WORK ACTION keywords.

Restrictions

You cannot alter the threshold type with the ALTER THRESHOLD statement. For example: You cannot change a TOTAL MEMBERCONNECTIONS threshold into a TOTALSCMEMBERCONNECTIONS threshold, for example. If you require a different threshold type, drop the existing thresholds and then create a new threshold.

Procedure

To alter a threshold:
1. Specify one or more of the following properties for the threshold on the ALTER THRESHOLD statement. You can change the following properties:
   • The boundary for the threshold predicate.
   • The actions to be taken, if the threshold boundary is violated.
   • Whether the threshold is enabled or disabled.
2. Commit your changes. When you commit your changes, the threshold is updated in the SYSCAT.THRESHOLDS view.

Dropping a threshold

Drop a threshold that you no longer require using the DDL statement DROP THRESHOLD.

Before you begin

To drop a threshold, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

If you want to drop a threshold in a work action set, use the ALTER WORK ACTION SET statement. You can also drop a threshold by dropping the entire WORK ACTION SET with the DROP statement.

Procedure

To drop a threshold:
1. Do one of the following steps:
   • If the threshold is a queuing threshold, use the ALTER THRESHOLD statement to disable it.
   • If you disabled a queuing threshold by using an ALTER THRESHOLD statement, issue a COMMIT statement to commit the change.
2. Use the DROP THRESHOLD statement to drop the threshold.
3. Commit your changes. When you commit your changes the threshold is removed from the SYSCAT.THRESHOLDS view.
Example: Using thresholds

You can use thresholds for a variety of purposes. In this scenario, thresholds are used to control the number of large jobs running in order to permit different execution times for different applications, and to control the behavior of an application that is in development.

You could use a DB2 workload management solution to divide and manage the database resources for the various departments in a company. For example, assume that the sales department runs two main reports, which consist of the monthly and yearly sales. Assume also that the human resources department runs a payroll application every other week and that the development team is working on a new type of report at the request of the management team. To define different execution environments for these departments, create service classes:

```sql
CREATE SERVICE CLASS SALES
CREATE SERVICE CLASS HUMANRESOURCES
CREATE SERVICE CLASS DEVELOPMENT
```

In this situation, you create a workload definition for each one of these applications to map the application to its applicable service superclass:

```sql
CREATE WORKLOAD MONTHLYSALES APPLNAME('monthlyrpt.exe') SERVICE CLASS SALES
CREATE WORKLOAD YEARLYSALES APPLNAME('yearlyrpt.exe') SERVICE CLASS SALES
CREATE WORKLOAD PAYROLL APPLNAME('payroll.exe') SERVICE CLASS HUMANRESOURCES
CREATE WORKLOAD NEWREPORT APPLNAME('dev.exe') SERVICE CLASS DEVELOPMENT
```

The database catalog therefore contains the following workload definitions:

- MonthlySales, mapping to the service superclass Sales
- YearlySales, mapping to the service superclass Sales
- Payroll, mapping to the service superclass Human Resources
- NewReport, mapping to the service superclass Development

Threshold on the number of large jobs

Because the YearlySales report is very large, you do not want to have more than one occurrence of this application running in the database at any time. You therefore create a threshold to set the maximum number of concurrent occurrences of this workload to 1:

```sql
CREATE THRESHOLD SINGLEYEARLYSALESRPT FOR WORKLOAD YEARLYSALES ACTIVITIES
ENFORCEMENT MEMBER
WHEN CONCURRENTWORKLOADOCCURRENCES > 1
STOP EXECUTION
```

You can achieve a similar solution by associating the YearlySales application with a service subclass YearlySalesReports (under the Sales service superclass) and setting the maximum concurrency threshold to a value of 1 for the service subclass:

```sql
CREATE SERVICE CLASS YEARLYSALESREPORTS UNDER SALES
ALTER WORKLOAD YEARLYSALES SERVICE CLASS YEARLYSALESREPORTS UNDER SALES

CREATE THRESHOLD SINGLEYEARLYSALESREPORT FOR SERVICE CLASS YEARLYSALESREPORTS
UNDER SALES ACTIVITIES ENFORCEMENT DATABASE
WHEN CONCURRENTDBCOORDACTIVITIES > 1
STOP EXECUTION
```

In either situation, you can set the threshold action to STOP EXECUTION to prevent more than one occurrence of the workload from executing. You can also
collect activity information if you want additional information about the conditions when the threshold is violated.

Threshold on activity lifetimes

Because all applications are expected to complete in an hour or less, you create a threshold with a database domain, preventing any activity from running longer than 1 hour. The only exception to this rule is the yearly report, which can take up to 5 hours to complete. Therefore, you can associate an activity total time threshold of 5 hours with the YearlySales workload. This will override the activity total time threshold applied to the yearly sales report, relaxing the time constraints. The new value of 5 hours now applies to the YearlySales workload although the global value of 1 hour applies elsewhere in the database:

CREATE THRESHOLD MAXDBACTIVITYTIME FOR DATABASE ACTIVITIES
  ENFORCEMENT DATABASE
  WHEN ACTIVITYTOTALTIME > 1 HOUR
  STOP EXECUTION

CREATE THRESHOLD MAXYRPTACTIVITYTIME FOR WORKLOAD YEARLYSALES
  ACTIVITIES ENFORCEMENT DATABASE
  WHEN ACTIVITYTOTALTIME > 5 HOURS
  STOP EXECUTION

Threshold on the number of coordinator and nested activities

The NewReport application makes heavy use of stored procedures and user-defined functions and is not fully debugged yet, so it tends to generate large numbers of activities that impact the rest of the system. After consulting with the developer, you learn that this new report is not supposed to generate more than 20 activities in total, so you define a threshold of type workload activities on the NewReport workload and set it to 20. Initially, you set the threshold action to STOP EXECUTION and COLLECT ALL to stop any unwanted side effect of the application starting large numbers of activities and to help the developer identify any problems:

CREATE THRESHOLD MAXDEVACTIVITIES FOR SERVICE CLASS DEVELOPMENT ACTIVITIES
  ENFORCEMENT DATABASE
  WHEN CONCURRENTDBCOORDACTIVITIES > 20
  COLLECT ACTIVITY DATA WITH DETAILS AND VALUES
  STOP EXECUTION

When the application becomes more stable, it enters its optimization phase. During the phase, the developer tries to reduce the number of activities generated by the application from between 15 and 20 to 15. At this time, you alter the threshold by changing its upper boundary value to 15 and the threshold action to CONTINUE. This threshold definition helps identify and address situations in which the number of generated activities exceeds 15 but the increased stability of the application does not require that its execution be stopped.

ALTER THRESHOLD MAXDEVACTIVITIES
  WHEN CONCURRENTDBCOORDACTIVITIES > 15
  COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS AND VALUES
  CONTINUE

Threshold to limit how long a unit of work runs

The application LongUOW issues transactions that can occasionally run longer than the desired ten minutes. This results in locks being held for too long and prevents more important applications from proceeding. In this case, you want to force the application, rather than let it hold up other work. You can restrict the
runtime for this application's transactions to an administrator-defined period of
time using the UOWTOTALTIME threshold.

First, create a workload for the LongUOW application:

```sql
CREATE WORKLOAD LONG_UOW APPLNAME('LONGUOW') SERVICE CLASS SYSDEFAULTUSERCLASS
```

Then, create a threshold for this workload that forces the LongUOW application
when any of the application's transactions take more than 10 minutes to finish:

```sql
CREATE THRESHOLD FORCELONGUOW FOR WORKLOAD LONG_UOW ACTIVITIES ENFORCEMENT DATABASE
WHEN UOWTOTALTIME > 10 MINUTES FORCE APPLICATION
```

You can also apply this threshold at the service subclass level or database level.

### Connection thresholds

A connection threshold applies controls to individual database connections. You
use connection thresholds to limit the total number of concurrent connections
to the database and how long a connection can sit idle.

**CONNECTIONIDLETIME threshold**
The CONNECTIONIDLETIME threshold specifies a maximum amount of time that
a connection can be idle (that is, not working on a user request).

<table>
<thead>
<tr>
<th>Type</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition domain</td>
<td>Database or service superclass</td>
</tr>
<tr>
<td>Enforcement scope</td>
<td>Database</td>
</tr>
<tr>
<td>Tracked work</td>
<td>User connections</td>
</tr>
</tbody>
</table>

If a connection remains idle for longer than the duration specified by the threshold
and the threshold action is STOP EXECUTION, the connection is closed.

### Activity thresholds

An activity threshold applies to an individual activity. When the resource usage of
an individual activity violates the upper bound of the threshold that is tracking it,
the corresponding action is triggered and applied once to the activity.

After being applied once, the threshold is deactivated for the activity and not
applied again.

For example: Assume that you defined a time based threshold that triggers a
CONTINUE action after an elapsed time of 5 minutes. If an activity violates this
threshold, the action is applied once but not reapplied every 5 minutes.
Domain precedence for activity thresholds

Activity thresholds apply to individual activities. If multiple thresholds apply to the same executing activity, a decision about which threshold to enforce must be made.

Aggregate thresholds are not affected because the same activity can contribute to multiple activity aggregates simultaneously, as occurs with concurrency thresholds, for example.

The resolution about which activity threshold to apply to an executing activity follows the rule that a value defined in a local domain overrides any value from a wider or more global domain. The hierarchy of domains is as follows, from the most local to the most global:

1. Statement
2. Work action (at workload level)
3. Workload
4. Service subclass
5. Service superclass
6. Work action (at database level)
7. Database

The following example shows how thresholds are overridden:

Examples

The following example shows how thresholds are overridden:

- A threshold that defines a maximum execution time of 1 hour for all database queries defined in the database domain is overridden by a threshold that defines a maximum execution time of 5 hours for a service superclass set up to handle large queries.
- That same service superclass threshold is overridden by a threshold for very large queries that defines the maximum execution time for a service subclass to be 10 hours.
- The maximum execution time of 1 hour defined in the database domain can be overridden by a value of 10 minutes in a second service superclass geared towards ensuring that shorter, important queries can complete quickly.
- The execution of a statement with text that matches the text that is specified for a threshold in the statement domain results in a threshold violation that overrides all other thresholds.

ACTIVITYTOTALTIME threshold

The ACTIVITYTOTALTIME threshold specifies the maximum amount of time that the data server can spend processing an activity.

Type Activity

Definition domain Database, service superclass, service subclass, work action, workload and statement

Enforcement scope Database

Tracked work Recognized coordinator and nested activities (see “Activities” on page 15)
Queuing
  No

Unit  Time duration expressed in seconds, minutes, hours, or days. If the time unit specified is seconds, this value must be a multiple of 10.

Predictive or reactive
  Reactive

This time represents the entire lifetime of the activity, which includes any time spent waiting on workload management queues, and any other wait times that are incurred while the activity is executing, such as lock wait times. When a cursor is opened, the activity associated with the cursor lasts until the cursor is closed. The activities that are covered by this threshold include the execution of SQL statements, not including compilation time, and the load utility.

When a time threshold is applied to a stored procedure, it also applies to work happening inside the stored procedure. Consequently, when a stored procedure time threshold expires, any work happening inside the stored procedure is stopped. Hierarchies of stored procedure invocations can lead to hierarchies of time thresholds being applied to activities executing in the innermost levels of nesting. The most restrictive time threshold in the hierarchy (that is, the time threshold with the closest deadline) is always the one that applies.

The data server considers IMPORT, EXPORT, and other CLP commands to be user logic. Activities that are invoked from within IMPORT, EXPORT, and other CLP commands are subject to thresholds.

CPUTIME threshold
The CPUTIME threshold specifies the maximum amount of combined user and system processor time that an activity can use on a particular member while the activity is running. Use this threshold to detect and control activities that are using excessive processor resources.

Type  Activity

Definition domain
  Database, service superclass, service subclass, work action, workload and statement

Enforcement scope
  Member

Tracked work
  See the information later in this topic

Queuing
  No

Unit  Time

Predictive or reactive
  Reactive

The amount of processor time that an activity spends running is measured from the time that the activity begins running at the member, after any queuing by thresholds, until the time that the activity finishes running.

Activities tracked by this threshold are as follows:
  • All DML activities.
CALL activities. The processor time for a CALL activity does not include the processor time of any child activity. The processor time spent in fenced processes is also not counted toward the total processor time for the CALL activity.

Activities that are initiated by the database manager through a utility or procedure, with the exception of the ADMIN_CMD procedure, are not counted for this condition. The data server considers IMPORT, EXPORT, and other CLP commands to be user logic. Activities that are invoked from within IMPORT, EXPORT, and other CLP commands are subject to thresholds. Child activities of the LOAD command are not tracked by this threshold.

**Example**

The following example creates a CPUTIME threshold TH1 for the database domain with a member enforcement scope. This threshold stops any activity that takes longer than 30 seconds to run, which it checks for at 5-second intervals. You can use this threshold to ensure that no queries on the system use an unreasonable amount of processor time, which can negatively impact other work running on the system.

```
CREATE THRESHOLD TH1 FOR DATABASE ACTIVITIES
  ENFORCEMENT MEMBER
  WHEN CPUTIME > 30 SECONDS CHECKING EVERY 5 SECONDS
  STOP EXECUTION;
```

**CPUTIMEINSC threshold**

The in-service-class CPUTIMEINSC threshold specifies the maximum amount of combined user and system processor time that an activity may use on a particular member while running in a specific service subclass. Use this threshold to detect and control activities that are using excessive processor resources.

<table>
<thead>
<tr>
<th>Class</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>domain</td>
</tr>
<tr>
<td>Service subclass</td>
<td></td>
</tr>
<tr>
<td>Enforcement scope</td>
<td>Member</td>
</tr>
<tr>
<td>Tracked work</td>
<td>See the information later in this topic</td>
</tr>
</tbody>
</table>

**Queueing**

No

**Unit**

Time

**Predictive or reactive**

Reactive

The processor time that an activity spends running is measured from the time that the activity enters the current service subclass until the time that the activity leaves the service subclass or finishes running.

This threshold differs from the CPUTIME threshold in that it controls only the amount of processor time that may be used in a specific service subclass, not the total amount of processor time used during the lifetime of the activity.

Activities tracked by this threshold are as follows:

- All DML activities.
CALL activities. Note that the processor time for a CALL activity does not include the processor time of any child activity. The processor time spent in fenced processes is also not counted towards the total processor time for the CALL activity.

Activities that are initiated by the database manager through a utility or procedure, with the exception of the ADMIN_CMD procedure, are not counted for this condition. The data server considers IMPORT, EXPORT, and other CLP commands to be user logic. Activities that are invoked from within IMPORT, EXPORT, and other CLP commands are subject to thresholds. Child activities of the LOAD command are not tracked by this threshold.

You can use the REMAP ACTIVITY action to control activities by remapping them to a service subclass with different resource assignments.

**Example**

The following example creates two service subclasses, A1 and A2, under a superclass A, with a single in-service-class CPUTIMEINSC threshold that remaps activities between subclasses after 1 minute of processor time has been used during query evaluation in service subclass A1. An event monitor record is logged.

```sql
CREATE SERVICE CLASS A;
CREATE SERVICE CLASS A1 UNDER A;
CREATE SERVICE CLASS A2 UNDER A;

CREATE THRESHOLD T1 FOR SERVICE CLASS A1 UNDER A
ACTIVITIES ENFORCEMENT MEMBER
WHEN CPUTIMEINSC > 1 MINUTE CHECKING EVERY 30 SECONDS
REMAP ACTIVITY TO A2 LOG EVENT MONITOR RECORD;
```

**DATATAGINSC threshold**

The in-service-class DATATAGINSC threshold checks the data tag value of a table space or storage group accessed by an activity. Use this threshold to dynamically map an activity to a different DB2 service subclass based on which data is being accessed.

**Type**  Activity

**Definition domain**  Service subclass

**Enforcement scope**  Member

**Tracked work**  Recognized coordinator

**Queuing**  No

**Unit**  Data tag

**Predictive or reactive**  Reactive

Data can be tagged with a numeric value by specifying the DATA TAG attribute when creating or altering a table space or a storage group. Data tag thresholds are evaluated when a scan is first opened on a table and when an insert is performed into a table. Any new data tag thresholds picked up by the activity after a scan has been opened as a result of a remap operation is not applied to that scan.
Only the data tags for the data table spaces being accessed in a query are considered by the threshold. The data tags values for index or long table spaces are not considered during a threshold evaluation. For example, if the query `SELECT COUNT(*) FROM T1` is index-only access with the index placed in a table space with data tag 3 and the table data placed in a table space with data tag 1, when the query is executed the threshold evaluation will use data tag 1 (the data tag for the data table space). The data tag threshold behavior is independent of the access method (index or direct scan) chosen in the query access plan.

Activities tracked by this threshold are:

- Coordinator activities of type DML and corresponding subagent work (subsection execution).
- Nested DML activities that are derived from user applications. Consequently, DML activities that are issued by the database manager, such as utilities, SYSPROC procedures (with the exception of SYSPROC.ADMIN_CMD) or internal SQL statements are unaffected by this threshold.

In the following usage scenario, the user creates 3 table spaces; TBHIGH contains high priority data and has a data tag of 1, TBMED contains medium priority data and has a data tag of 4, and TBLOW contains low priority data and has a data tag of 9. The activities are initially mapped to a high priority service subclass. If any activities touch data on a table space other than that on the TBHIGH table space, the user wants the priority of those activities to drop. In the following example, a service superclass MAINSC containing three service subclasses, SCHIGH, SCMED, and SCLOW (for executing high, medium, and low priority work) have already been created.

Create a DATATAGINSC threshold on the high priority service subclass, and if any activity touches any data with a data tag of anything other than 1, map it to the medium priority service subclass.

```
CREATE THRESHOLD MAPTOMED FOR SERVICE CLASS SCHIGH UNDER MAINSC ACTIVITIES
ENFORCEMENT MEMBER WHEN DATATAGINSC NOT IN (1) REMAP ACTIVITY TO SCMED
```

Create a SQLDATATAGINSC threshold on the medium priority service subclass, and if any activity touches any data with a data tag of 9 map it to the low priority service subclass.

```
CREATE THRESHOLD MAPTOLOW FOR SERVICE CLASS SCMED UNDER MAINSC ACTIVITIES
ENFORCEMENT MEMBER WHEN DATATAGINSC IN (9) REMAP ACTIVITY TO SCLOW
```

If an activity touches a table space with a data tag of 9, that activity will be remapped to SCMED, and then immediately remapped again to SCLOW (due to the threshold on the SCMED service subclass). The user in this case, will see two threshold violations.

**ESTIMATEDSQLCOST threshold**

The ESTIMATEDSQLCOST threshold specifies the maximum estimated cost that is permitted for DML activities.

**Type** Activity

**Definition domain**

- Database, service superclass, service subclass, work action, workload, and statement

**Enforcement scope**

- Database
Tracked work

See the information later in this topic

Queuing

No

Unit

Estimated SQL cost expressed in timerons

Predictive or reactive

Predictive

This threshold tracks the following activities:

- DML activities that are issued at the coordinator member.
- Nested DML activities that are invoked from a user application. Consequently, DML activities that are issued by the data server internally, such as DML activities issued from within the DB2 utilities, SYSPROC stored procedures, and internal SQL, are unaffected by this threshold unless their cost is included in the parent activity estimate. In this situation, these activities are indirectly tracked. A trigger is an example of an indirectly tracked activity. IMPORT, EXPORT, and other CLP commands are considered to be user logic. Activities that are invoked from within IMPORT, EXPORT, and other CLP commands are subject to thresholds. For information about the activities that fall under a work class with the DML work type, see “Work identification by type, cost, or data accessed with work classes” on page 48.

SQLROWSREAD threshold

The SQLROWSREAD threshold specifies the maximum number of rows that a DML activity may read on any member. Use this threshold to detect and control activities that are reading an excessive number of rows.

Class

Activity

Definition domain

Database, service superclass, service subclass, work action, workload and statement

Enforcement scope

Member

Tracked work

See the information later in this topic

Queuing

No

Unit

Number of rows

Predictive or reactive

Reactive

This threshold differs from the SQLROWSRETURNED threshold in that it controls the maximum number of rows read during query evaluation, not the number of rows returned to a client application from the data server.

Index accesses are not counted toward the total number of rows read. If an access plan uses only indexes during query evaluation, the SQLROWSREAD threshold will not be violated.

This threshold is evaluated at user-configurable time intervals; if the interval is greater than the amount of time it takes to exceed the number of rows read, it is
possible for the number of rows read for an activity on a member to exceed the threshold boundary, before the violation is detected.

Activities tracked by this threshold are as follows:

- Coordinator activities of type DML and corresponding subagent work such as subsection execution.
- Nested DML activities that are derived from user applications. Consequently, DML activities that are issued by DB2 logic, such as utilities, SYSPROC procedures, or internal SQL statements, are unaffected by this threshold. IMPORT, EXPORT, and other CLP commands are considered to be user logic; therefore, activities that are invoked from within IMPORT, EXPORT, and other CLP commands are subject to thresholds.

**Example**

The following example creates an SQLROWSREAD threshold TH1 for the database domain with a member enforcement scope. This threshold stops the execution of any activity that reads more than 5,000,000 rows during query evaluation, which the threshold checks for at 10-second intervals. You can use this threshold to ensure that no queries on the system read an unreasonable number of rows, which can negatively impact other work running on the system.

```sql
CREATE THRESHOLD TH1 FOR DATABASE ACTIVITIES
   ENFORCEMENT MEMBER
   WHEN SQLROWSREAD > 5000000 CHECKING EVERY 10 SECONDS
   STOP EXECUTION;
```

**SQLROWSREADINSC threshold**

The in-service-class SQLROWSREADINSC threshold specifies the maximum number of rows that a DML activity can read on a particular member while running in a specific service subclass. Use this threshold to detect and control activities that are reading an excessive number of rows.

<table>
<thead>
<tr>
<th>Class</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition domain</td>
<td>Service subclass</td>
</tr>
<tr>
<td>Enforcement scope</td>
<td>Member</td>
</tr>
<tr>
<td>Tracked work</td>
<td>See the information later in this topic</td>
</tr>
<tr>
<td>Queuing</td>
<td>No</td>
</tr>
<tr>
<td>Unit</td>
<td>Number of rows</td>
</tr>
<tr>
<td>Predictive or reactive</td>
<td>Reactive</td>
</tr>
</tbody>
</table>

This threshold differs from the SQLROWSREAD threshold in that it controls the number of rows read only from the time that an activity enters a specific service subclass, not the total number of rows read during the lifetime of the activity. This threshold also differs from the SQLROWSRETURNED threshold in that it controls the maximum number of rows read during query evaluation in the current service subclass, not the number of rows returned to a client application from the data server.
Index accesses are not counted toward the total number of rows read. If an access plan uses only indexes during query evaluation, the SQLROWSREADINSC threshold will not be violated.

This threshold is evaluated at user-configurable time intervals; if the interval is greater than the amount of time it takes to exceed the number of rows read, it is possible for the number of rows read for an activity on a member to exceed the threshold boundary, before the violation is detected.

Activities tracked by this threshold are as follows:
- Coordinator activities of type DML and corresponding subagent work such as subsection execution.
- Nested DML activities that are derived from user applications. Consequently, DML activities that are issued by DB2 logic, such as utilities, SYSPROC procedures, or internal SQL statements, are unaffected by this threshold. IMPORT, EXPORT, and other CLP commands are considered to be user logic; therefore, activities that are invoked from within IMPORT, EXPORT, and other CLP commands are subject to thresholds.

You can use the REMAP ACTIVITY action to control activities by remapping them to a service subclass with different resource assignments.

**Example**

The following example creates two service subclasses, A1 and A2, under a superclass A, with a single in-service-class SQLROWSREADINSC threshold that remaps activities between subclasses after 10 000 rows have been read in service subclass A1 during query evaluation. An event monitor record is logged.

```sql
CREATE SERVICE CLASS A;
CREATE SERVICE CLASS A1 UNDER A;
CREATE SERVICE CLASS A2 UNDER A;
CREATE THRESHOLD T1 FOR SERVICE CLASS A1 UNDER A
  ACTIVITIES ENFORCEMENT MEMBER
  WHEN SQLROWSREADINSC > 10000 REMAP ACTIVITY TO A2
  LOG EVENT MONITOR RECORD;
```

**SQLROWSRETURNED threshold**

The SQLROWSRETURNED threshold specifies the maximum number of rows that can be returned by the data server to the client.

<table>
<thead>
<tr>
<th>Type</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition domain</td>
<td>Database, service superclass, service subclass, work action, workload and statement</td>
</tr>
<tr>
<td>Enforcement scope</td>
<td>Database</td>
</tr>
<tr>
<td>Tracked work</td>
<td>See the information later in this topic</td>
</tr>
<tr>
<td>Queuing</td>
<td>No</td>
</tr>
<tr>
<td>Unit</td>
<td>Number of rows</td>
</tr>
<tr>
<td>Predictive or reactive</td>
<td>Reactive</td>
</tr>
</tbody>
</table>
When multiple result sets are returned by a CALL statement, the threshold applies
to each result set separately and not as an aggregate to the total number of rows
returned across all result sets. For example, if you define the threshold for 20 rows
and the CALL statement returns two result sets returning 15 rows and 19 rows
respectively, the threshold is not triggered.

Activities tracked by this threshold are as follows:
- DML activities that are issued at the coordinator member.
- Nested DML activities that are invoked from a user application. Consequently,
  DML activities that are issued by the data server internally, such as DML
  activities issued from within the DB2 utilities, SYSPROC stored procedures, and
  internal SQL, are unaffected by this threshold.

**SQLTEMPSPACE threshold**
The SQLTEMPSPACE threshold specifies the maximum amount of system
temporary table space that can be consumed by a DML activity at any member.
DML activities often use temporary table space for operations such as sorting and
the manipulation of intermediate result sets.

<table>
<thead>
<tr>
<th>Type</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition domain</td>
<td>Database, service superclass, service subclass, work action, workload and statement</td>
</tr>
<tr>
<td>Enforcement scope</td>
<td>Member</td>
</tr>
<tr>
<td>Tracked work</td>
<td>See the information later in this topic</td>
</tr>
<tr>
<td>Queuing</td>
<td>No</td>
</tr>
<tr>
<td>Unit</td>
<td>Amount of temporary table space expressed in kilobytes (KB), megabytes (MB), or gigabytes (GB)</td>
</tr>
<tr>
<td>Predictive or reactive</td>
<td>Reactive</td>
</tr>
</tbody>
</table>

Activities tracked by this threshold are as follows:
- DML activities that are issued at the coordinator member.
- Nested DML activities that are derived from user applications. Consequently,
  DML activities that are issued by DB2 logic (such as utilities, SYSPROC
  procedures, or internal SQL) are unaffected by this threshold.

The data server considers IMPORT, EXPORT, and other CLP commands to be user
logic. Activities that are invoked from within IMPORT, EXPORT, and other CLP
commands are subject to thresholds.

**Aggregate thresholds**
An aggregate threshold places collective control over elements of work in a
database. The boundary that you define using an aggregate threshold operates as a
running total, to which any work tracked by the threshold contributes.

When newly instantiated work causes the upper boundary to be violated, the
respective action is triggered. The work that caused the upper boundary to be
violated is the only one affected by the triggered action.
Activity queuing

Some thresholds have a built-in queue and permit you to enforce how many activities can execute concurrently by queuing all additional activities once the concurrency limit is reached, up until the set limit for the queue is exceeded.

When the number of activities exceeds the concurrency limit for a queuing threshold, new requests are queued automatically in a first-in, first-out fashion, until the queue reaches the size specified by the queuing boundary of the threshold definition. When the queue is full, the upper boundary is reached. The next request is considered to have violated the threshold, and the action specified for the threshold is applied to it. For example, an action of STOP EXECUTION causes the newly arrived work to be rejected.

You can also define the upper queuing boundary as unbounded, in which case there is no upper limit to the size of the queue. In this situation, when newly arriving work is added to the queue, the threshold is not considered to have been violated, regardless of how large the queue has become. If you define a hard limit for the upper boundary of the queue and define CONTINUE as the threshold violation action, all newly arriving work that exceeds the hard limit is considered to have violated the threshold; a threshold violation is created but the new work is still added to the queue.

AGGSQLTEMPSPACE threshold

The AGGSQLTEMPSPACE threshold specifies the maximum amount of system temporary table space that can be used in total across all concurrently running DML activities in a service subclass. DML activities often use temporary table space for operations such as sorting and the manipulation of intermediate result sets.

Class Aggregate
Definition domain Service subclass
Enforcement scope Member
Tracked work See the information later in this topic
Queuing No
Unit Kilobytes, megabytes, or gigabytes
Predictive or reactive Reactive

Activities tracked by this threshold are as follows:
- DML activities that are issued at the coordinator member.
- Nested DML activities that are derived from user applications. Consequently, DML activities that are issued by DB2 logic, such as utilities, SYSPROC procedures, or internal SQL statements, are unaffected by this threshold. IMPORT, EXPORT, and other CLP commands are considered to be user logic; therefore, activities that are invoked from within IMPORT, EXPORT, and other CLP commands are subject to thresholds.
**CONCURRENTDBCOORDACTIVITIES** threshold

The CONCURRENTDBCOORDACTIVITIES threshold specifies the maximum number of recognized coordinator activities that can run concurrently in the specified definition domain and enforcement scope.

The use of this type of threshold is best suited for applications that do not execute more than one activity at a time. If an application starts more than one activity concurrently, such as issuing an UPDATE SQL statement while a cursor is open, then certain queue contention scenarios can sometimes occur depending on the concurrency level allowed by the threshold and the behaviors of the other applications involved. If this threshold exists in scenarios where applications can execute more than one activity concurrently or the application behavior is unknown, then it is recommended to have an ACTIVITYTOTALTIME threshold defined for those activities to help automatically resolve any potential queue contention scenarios.

**Type**  Aggregate

**Definition domain**  
Database, work action, service superclass, service subclass

**Enforcement scope**  
Database in environments other than DB2 pureScale environments
Member in a DB2 pureScale environment

**Tracked work**  
Recognized coordinator and nested activities (see “Work identification by type, cost, or data accessed with work classes” on page 48)

**Queuing**  
Yes

**Unit**  Number of concurrent database activities

**Predictive or reactive**  
Predictive

This threshold is a generalization of the CONCURRENTWORKLOADACTIVITIES threshold. The CONCURRENTWORKLOADACTIVITIES applies only to activities that are running in a workload domain, but you can apply the CONCURRENTDBCOORDACTIVITIES threshold to a variety of domains, ranging from the entire database to a single work action. The CONCURRENTDBCOORDACTIVITIES threshold tracks coordinator activities for non-CALL statements and any nested activities generated by CALL statements. Unlike the CONCURRENTWORKLOADACTIVITIES threshold, the CONCURRENTDBCOORDACTIVITIES threshold is a queuing threshold.

**Note:** If a threshold action of CONTINUE is specified for a queuing threshold, it effectively makes the size of the queue unbounded, regardless of any hard value specified for the queue size. Use the `concurrent_act_top` monitor element to determine the highest concurrency of activities (including nested activities) that was reached on a member for a service subclass in the time interval collected.

When creating queuing thresholds of the CONCURRENTDBCOORDACTIVITIES type, be aware of configurations that might lead to irreconcilable queue-based contention. For example:

1. A concurrency threshold of type CONCURRENTDBCOORDACTIVITIES is created with a maximum concurrency value of 1 and a queue size greater than 1.
2. An application opens a cursor that the DB2 data server recognizes as activity A1, which consumes the unique ticket that is available for the threshold.

3. While the cursor is still open, the application issues an UPDATE statement, which the data server recognizes as activity A2, and which is also subject to the concurrency threshold. Because the A1 activity is already running, the new activity A2 is queued.

   The application is now in a queue contention state that cannot be resolved. It is waiting for A2 to execute but A2 is waiting for A1 to finish executing. This situation will not resolve itself without external intervention unless an ACTIVITYTOTALTIME threshold is also defined over these activities. If ACTIVITYTOTALTIME threshold is defined, then the situation will be resolved when A1 violates that threshold by spending more than the maximum amount of time that the data server should spend processing that activity.

   This example can be generalized to multiple applications and queues. You can resolve this situation by increasing the concurrency values, or cancelling certain activities if the concurrency values are correctly set.

   To reduce the chance of creating irreconcilable queue contention scenarios, the CONCURRENTDBCOORDACTIVITIES threshold affects different types of activities as follows:
   - CALL statements are not controlled by the threshold, but all nested child activities are under threshold control. Note that both anonymous blocks and autonomous routines are classified as CALL statements.
   - User defined functions (UDFs) are under threshold control, but child activities nested within UDFs are not controlled. If an autonomous routine is called from within a user defined function, neither the autonomous routine and nor any child activities of the autonomous routine are under threshold control.
   - Trigger actions that invoke CALL statements and the child activities of these CALL statements are not under threshold control. Note that the INSERT, UPDATE, and DELETE statements themselves that can cause a trigger to activate are under threshold control.

**CONCURRENTWORKLOADACTIVITIES threshold**

The CONCURRENTWORKLOADACTIVITIES threshold specifies the maximum number of coordinator and nested activities that can concurrently run in a workload occurrence.

- **Type** Aggregate
- **Definition domain** Workload
- **Enforcement scope** Workload occurrence
- **Tracked work** Recognized coordinator and nested activities (see “Activities” on page 15)
- **Queuing** No
- **Unit** Number of concurrent workload activities
- **Predictive or reactive** Predictive
This threshold applies to a single workload occurrence. If you have multiple occurrences of a workload running concurrently, the threshold applies separately to each workload occurrence. The tracked activities include all recognized coordinator activities and any nested activities that are generated as a result of the execution of the coordinator activity. For example, unlike the CONCURRENTDBCOORDACTIVITIES threshold, if a stored procedure is called and that stored procedure executes some SQL, both the CALL statement (which is the coordinator activity) and the SQL statements executed by the stored procedure (which are the nested activities) count towards the threshold total.

COMMIT, ROLLBACK, and ROLLBACK to SAVEPOINT statements are unaffected by this threshold.

Nested activity considerations

The nested activities that are tracked by this threshold must satisfy the following criteria:

- They must be a recognized coordinator activity. Nested coordinator activities that are not recognized types as described in “Work identification by type, cost, or data accessed with work classes” on page 48 are not counted.
- They must be directly invoked from user logic, such as a user-written stored procedure issuing SQL or from the SYSPROC.ADMIN_CMD stored procedure. Nested coordinator activities that are started by the invocation of a DB2 utility or any other code in the SYSIBM, SYSFUN, or SYSPROC schemas are not counted towards the upper boundary specified by this threshold.

Example

In this example, the CONCURRENTWORKLOADACTIVITIES threshold maximum value is set to 5. The user logic causes the following sequence of operations to occur in a workload occurrence:

1. Issue a load command: the current number of workload activities is 1.
   - The load command internally issues some SQL. The current number of workload activities is 1. (SQL generated by a utility does not count against the CONCURRENTWORKLOADACTIVITIES threshold.)
   - The load command ends. The current number of workload activities is 0.
2. CALL the SYSPROC.SP1 stored procedure. The current number of workload activities is 1.
   - The SYSPROC.SP1 stored procedure generates some SQL. The current number of workload activities is 1. (SQL generated by a utility does not count against the CONCURRENTWORKLOADACTIVITIES threshold.)
   - The SYSPROC.SP1 stored procedure ends. The current number of workload activities is 0.
3. Open a cursor C1. The current number of workload activities is 1.
4. Issue a runstats command. The current number of workload activities is 1.
   - The runstats command generates some SQL. The current number of workload activities is 1.
   - The runstats command ends. The current number of workload activities is 1.
5. Close the cursor C1. The current number of workload activities is 0.
6. CALL the BOB.SP1 stored procedure. The current number of workload activities is 1.
• The BOB.SP1 stored procedure opens three cursors. The current number of workflow activities is 4.
• The BOB.SP1 stored procedure calls the SYSPROC.SP2 stored procedure. The current number of workflow activities is 5.
  – The SYSPROC.SP2 stored procedure issues some SQL. The current number of workflow activities is 5.
  – The SYSPROC.SP2 stored procedure ends. The current number of workflow activities is 4.
• The BOB.SP1 stored procedure calls the BOB.SP2 stored procedure. The current number of workflow activities is 5.
  – The BOB.SP2 stored procedure issues some SQL. At this point, the threshold is triggered.
  – The BOB.SP2 stored procedure ends. The current number of workflow activities is 4.
• The BOB.SP1 stored procedure ends. The current number of workflow activities is 0.

7. Open a cursor C2. The current number of workflow activities is 1.
8. CALL the BOB.SP2 stored procedure. The current number of workflow activities is 2.

**CONCURRENTWORKLOADOCCURRENCES threshold**

The CONCURRENTWORKLOADOCCURRENCES threshold is an aggregate threshold that specifies the maximum number of workload occurrences that can run concurrently on the coordinator member.

**Type**  Aggregate

**Definition domain**  Workload

**Enforcement scope**  Member

**Tracked work**  Workload occurrences

**Queuing**  No

**Unit**  Number of concurrent workload occurrences

**Predictive or reactive**  Predictive

When a workload occurrence is started, if the work that it generates is sent to non-coordinator members, the work on these members does not count towards the concurrent threshold total on the coordinator member. For example, assume that a CONCURRENTWORKLOADOCCURRENCES threshold is defined to permit only one occurrence of workload A on a member. Then assume that an application connects to member 1, resulting in an occurrence of workload A being started, and that this workload causes work to be sent to database members 1, 2, and 3. In this situation, the total number of occurrences of workload A is one on member 1 and zero on database members 2 and 3. Therefore, if another application connects to member 1 and another occurrence of workload A is started on member 1, that workload is rejected. However, new occurrences of workload A can still be started on database members 2 and 3.
TOTALMEMBERCONNECTIONS threshold

The TOTALMEMBERCONNECTIONS threshold specifies the maximum number of concurrent database connections on a coordinator member for a database, that is, this threshold controls the maximum number of clients that can connect to the database on each of its database members.

This threshold is not enforced for users with DBADM and WLMADM authority.

**Type**  Aggregate  
**Definition domain**  Database  
**Enforcement scope**  Member  
**Tracked work**  Connections  
**Queueing**  Yes (enforced at 0)  
**Unit**  Number of concurrent connections  
**Predictive or reactive**  Predictive

For example, if you set the TOTALMEMBERCONNECTIONS threshold to 10 and the database has five members, each member can have up to 10 clients connected concurrently, for a total of 50 client connections across the entire database.

The TOTALMEMBERCONNECTIONS threshold controls only coordinator connections. Connections made by subagents are not counted towards the threshold.

This threshold is useful for situations in which you want to have multiple databases in the same instance. Setting a TOTALMEMBERCONNECTIONS threshold on a member ensures that client connections from one database cannot use all of the available connections on a member.

Ensure that you set the `max_connections` database manager configuration parameter high enough to support the maximum number of connections that you expect across the database. If you set a TOTALMEMBERCONNECTIONS threshold for a database, you must set `max_connections` to at least the threshold value. If you want to run multiple databases on the same instance, ensure that you set `max_connections` high enough to support the maximum number of connections for all databases. The data server does not check for this condition because it is impossible to know beforehand how many of the databases will be active concurrently.

**Note:** Although the queue size is typically enforced at 0 for the TOTALMEMBERCONNECTIONS threshold, if a threshold action of CONTINUE is specified for this queuing threshold, it effectively makes the size of the queue unbounded, regardless of any hard value specified for the queue size.

TOTALSCMEMBERCONNECTIONS threshold

The TOTALSCMEMBERCONNECTIONS threshold specifies the maximum number of concurrent database connections on a coordinator member for a service superclass.
Type   Aggregate

Definition domain
   Service superclass

Enforcement scope
   Member

Tracked work
   Connections

Queuing
   Yes

Unit
   Number of concurrent connections in service class

Predictive or reactive
   Predictive

When the TOTALSCMEMBERCONNECTIONS threshold in the service class is reached, subsequent coordinator connections that join the service superclass are queued until the specified queue size is reached. By default, the queue size is zero, which means that no connections can be queued. If a connection joins the queue of a TOTALSCMEMBERCONNECTIONS threshold, the connection is considered to be in a transient state.

Note: If a threshold action of CONTINUE is specified for a queuing threshold, it effectively makes the size of the queue unbounded, regardless of any hard value specified for the queue size.

Tracked connections include both new client connections and existing client connections that switch to the service class from another service class. Connections switch service classes by associating with a different workload definition that is mapped to a different service class. Workload reevaluation occurs only at transaction boundaries, so connections can switch service classes only at transaction boundaries; however, because resources that are associated with WITH HOLD cursors are maintained across transaction boundaries, connections with open WITH HOLD cursors cannot switch service superclasses. When the connection concentrator is on, any application that is switched leaves the service class. When the application is switched in at the subsequent statement, it must rejoin the service class and consequently pass the threshold.

When the queue size threshold is reached, the threshold action is triggered. The TOTALSCMEMBERCONNECTIONS threshold controls only coordinator connections. Connections made by subagents are not counted towards the threshold.

If you set a threshold value for TOTALMEMBERCONNECTIONS, set it large enough to accommodate the threshold that you specify for TOTALSCMEMBERCONNECTIONS. For example, if you define five service superclasses for a database and each of them has a TOTALSCMEMBERCONNECTIONS threshold value of 10, the TOTALMEMBERCONNECTIONS threshold value should be at least 50.

**Unit of work thresholds**

A unit of work threshold applies controls to individual units of work. When the resource usage of an individual unit of work violates the upper bound of the
threshold that is tracking it, the corresponding action is triggered and applied once to either the application (in the case of force application) or the unit of work (in the case of rollback).

**UOWTOTALTIME threshold**
The UOWTOTALTIME threshold specifies the maximum amount of time that a unit of work may spend in the DB2 engine.

**Type**  Unit of work

**Definition domain**  
Database, workload, service superclass

**Enforcement scope**  
Database

**Tracked work**  
See the information later in this topic.

**Queuing**  
No

**Unit**  Time duration expressed in seconds, minutes, hours, or days. If the time unit specified is seconds, this value must be a multiple of 10.

**Predictive or reactive**  
Reactive

The STOP EXECUTION action for a UOWTOTALTIME threshold rolls back the unit of work. The FORCE APPLICATION action forces the application to which the unit of work belongs. The COLLECT ACTIVITY DATA option can be specified for this threshold, but it is ignored.

A UOWTOTALTIME threshold defined on the workload domain overrides any UOWTOTALTIME threshold defined on the service superclass domain. A UOWTOTALTIME threshold defined on the service superclass domain overrides any UOWTOTALTIME threshold defined on the database domain.

**Example**

The following example creates a threshold FORCE10MINUTEUOW that issues a FORCE APPLICATION command when a unit of work has been executing for more than 10 minutes.

```
CREATE THRESHOLD FORCE10MINUTEUOW FOR DATABASE ACTIVITIES
  ENFORCEMENT DATABASE
  WHEN UOWTOTALTIME > 10 MINUTES
  FORCE APPLICATION;
```

---

**Priority aging of ongoing work**

Priority aging is an approach to workload management in which the priority of in-progress activities automatically changes over time.

The longer an activity runs, the lower its priority, and the fewer resources it will receive. You can use priority aging to control longer-running activities, so that throughput for shorter-running activities can be improved. The priority aging approach works when resource control responds to the movement of work between service classes. That is, when work that is already being processed changes service classes, the movement is actually reflected in the resources that are received by that work (as well as other work in the new service class). This
approach is best implemented when using the explicit CPU controls that are provided by the CPU dispatcher or by integration with operating system workload management products.

**Changing the priority of activities by remapping**

System resources are allocated and controlled by using service classes. With priority aging, the priority of an activity can be changed by moving the activity from one service class to another service class. The priority increases if the new service class has more resources, and the priority decreases if the new service class has fewer resources. Activities are moved when a threshold with a REMAPACTIVITY action is violated, based upon predetermined maximum usage of a specific resource such as processor time or rows read. After an activity is mapped to a new service class, it continues to run with the new resource constraints applied.

A simple approach that you can use to help short queries to run faster is to define a series of service classes with successively lower levels of resource priority and threshold actions that move activities between the service subclasses. Using this setup, you can decrease, or age, the priority of longer-running work over time and perhaps improve response times for shorter-running work without having detailed knowledge of the activities running on your data server.

You can create this setup by assigning a high priority for all applicable resources to one service class, medium priority to a second service class, and low priority to a third service class. As work enters the system, it is automatically placed into the first service class and begins running using the high-priority settings of this service class. If you also define thresholds for each of the service classes that limit the time or resources used during execution, work is dynamically reassigned to the next-lower service class if the threshold of the next-higher class is violated. This dynamic resource control is repeatedly applied until the work is completed or is in the lowest-priority class, where it remains until it is completed or you force it to stop running.

![Diagram showing three service classes with successively lower priority](image-url)
In-service-class thresholds

Remapping of activities is available with any of the in-service-class thresholds, which control the amount of a resource that may be used while an activity is running in a particular service subclass. Examples of resources are the amount of processor time used (CPUTIMEINSC threshold), and the number of rows read by an application (SQLROWSREADINSC threshold), per activity per member. These thresholds differ from other activity thresholds, which control resources used throughout the entire lifetime of an activity.

Because of the control that in-service-class thresholds provide over service subclasses, you can define in-service-class thresholds only on a service subclass domain. The in-service-class thresholds provide controls similar to DB2 Governor rules, which act on processor time and rows read monitor elements.

When an in-service-class threshold is associated with a REMAP ACTIVITY action, agents working for the activity periodically check whether the threshold has been violated on each member. If an agent detects a threshold violation on a member, the agent triggers the REMAP ACTIVITY action for the activity on the member and then remaps itself to the target service subclass. All other agents working for the activity on the same member remap themselves to the target service subclass when they detect that the activity has been remapped. Only one agent detects the threshold violation and remaps the activity, and the activity is considered remapped after that agent has detected the threshold violation and performed the remapping.

Two monitor elements provide information about activity remapping within service subclasses. The `act_remapped_in` monitor element provides a counter that records how many activities were remapped into a service subclass and is incremented each time for the target service subclass that an activity is remapped to. Similarly, the `act_remapped_out` monitor element counter is incremented each time for the source service subclass that an activity is remapped out of. An additional monitor element, `num_remaps`, counts the number of times in total that an activity has been remapped between service subclasses.

An activity can be remapped multiple times to different service subclasses, and an activity can return to its original service subclass after being remapped to another service subclass.

The in-service class thresholds are evaluated separately for an activity on each member, without coordination. Because there is no coordination between members, when an activity is remapped on one member, it is possible for the same activity to be in different service subclasses on different members simultaneously.

When subagent work for an activity is completed on a remote member and further work for the same activity is sent to the same member later, the activity restarts in the same service subclass as the agent that sent the request to the member. If you defined an in-service-class threshold for this service subclass, the timer or counter for the activity on the remote member restarts at zero.

Where activities are nested, parent and child activities are tracked separately. Therefore, if a child activity is using an excessive amount of resources, only this activity, not its parent or sibling activities, violates a threshold.
Using the in-service-class thresholds

On data servers where the primary resource activities have to compete for is processor time, use the CPUTIMEINSC threshold as your first measure of control. On data servers where queries reading many table rows result primarily in I/O contention, use SQLROWSREADINSC. On systems that see a combination of heavy processor and IO activity, use a combination of the CPUTIMEINSC and SQLROWSREADINSC thresholds.

How much of a given resource you permit activities to consume in a service subclass before remapping them to a different service subclass depends largely on your particular environment. To find the best value for each threshold condition, you need to monitor how activities are being processed on your data server. If the maximum amount of processor time that can be used or the maximum number of rows that can be read in a service class is set too high, activities will inappropriately start and finish in the same service subclass regardless of how much resource each activity requires. If the maximum processor time or rows read is set too low, no activity will finish in the service class it is originally mapped to and every activity will end up being remapped to the another service class regardless of business priority. In either case, your tiered configuration will not benefit the overall throughput on your data server and activities are not treated according to their business priority effectively.

In addition to determining how much of a given resource an activity can consume, some thresholds allow you to define a check interval for how often the data server checks for threshold violations. This capability is provided for thresholds where it is too expensive to check the threshold each time a unit of the resource being controlled is consumed and determines the latency with which violations of these thresholds are detected. The CPUTIME and SQLROWSREAD thresholds and their in-service class counterparts CPUTIMEINSC and SQLROWSREADINSC support check intervals. On serial database instances, the check interval equals the amount of real time that you want to elapse between checks for a threshold violation. In multimember database environments or on SMP instances, the check interval should be set to a value that is less than the amount of real time elapsed to take into account that there can be more than one agent accumulating processor time simultaneously for the activity. To calculate the approximate check interval in multimember database environments or on SMP instances, divide the amount of real time you want to elapse between checks by the degree of parallelism for the activity and use the resulting value for the CHECKING EVERY clause.

For example: In a single member database, if you want a CPUTIMEINSC threshold to trigger a REMAP ACTIVITY action after 30 seconds of processor time have been consumed, you can set the check interval to 30 seconds and be certain that the threshold action will be triggered after no more than 30 seconds of processor time have been consumed (processor time used cannot outstrip real time elapsed). In a multimember database environment, if you define a CPUTIMEINSC threshold that is set at 5 seconds with a check interval of 5 seconds, and an activity has 1 coordinator member agent and 4 subagents working on its behalf, it is possible for the activity to consume 5 seconds of CPU time in just 1 second of real time, because 5 agents simultaneously accumulate 1 second of processor time each. To prevent the activity from consuming a multiple of 5 seconds of processor time, the check interval should in this case be set to 1 second.

For additional information on how to use the thresholds, see the sample tiering scripts and priority aging scenarios.
Effect of remapping on thresholds

Which thresholds continue to apply after remapping through a REMAP ACTIVITY action depends on whether the thresholds apply only to a specific service subclass or throughout the lifetime of an activity.

When you remap an activity to a new service subclass, only the in-service-class thresholds, such as CPUTIMEINSC and SQLROWSREADINSC, change. These in-service-class thresholds no longer affect an activity after it leaves the source service subclass, and they are replaced with the corresponding thresholds for the target subclass, if you defined those thresholds. All other activity thresholds from the service subclass to which the activity was originally mapped remain unchanged, and applicable threshold timers and counters are not reset. The activity is not re-evaluated against any other thresholds that you defined for the target service subclass.

For example, assume that two service subclasses with thresholds are defined as follows:

- Service subclass A with the following thresholds:
  - An ACTIVITYTOTALTIME lifetime threshold TH1 with a STOP EXECUTION action after 30 minutes have elapsed
  - An SQLROWSREADINSC in-service-class threshold TH2 with a REMAP ACTIVITY action to service subclass B after more than 2000 rows have been read

- Service subclass B with the following thresholds:
  - An ACTIVITYTOTALTIME lifetime threshold TH3 with a STOP EXECUTION action after 5 minutes have elapsed
  - An SQLROWSREADINSC threshold TH4 with a STOP EXECUTION action after more than 1000 rows have been read

When an activity enters the system in service subclass A, both thresholds TH1 and TH2 apply to the activity. If the activity reads more than 2000 rows during query evaluation, it is dynamically remapped to service subclass B. Because of the remapping of the activity to subclass B, the applicable in-service-class thresholds change, and TH4 rather than TH2 now applies to the activity. Counters for both thresholds are reset to zero, and even though the activity has read more than 2000 rows in the original service subclass, the counter for TH4 is restarted at zero; the activity must read more than 1000 rows while running in service subclass B before threshold TH4 is violated. Threshold TH1, which applies throughout the lifetime of the activity, continues to apply, even though the activity is now running in a different subclass. Threshold TH3 does not exercise any control over the remapped activity at all, because it did not apply to the first service subclass that the activity entered when it began running.

Sample priority aging scripts

Use the provided sample scripts to quickly create a tiered service class configuration on your data server. With a tiered configuration, you can address specific performance goals by decreasing the priority of longer-running queries over time, known as priority aging. You can also modify the scripts according to your own business priorities when adapting them to your environment.

The two sample scripts wlm tiersdefault.db2 and wlm tierstimerons.db2 are intended to demonstrate how you might use priority aging on your data server to improve overall throughput. DB2 workload manager provides you with the
controls that can help with overall throughput on your data server, but to gain the full benefit of the scripts, and of priority aging in general, you will need to run your data server for an extended period of time and monitor how this work is performed, then adjust the service class and threshold settings accordingly.

The scripts are provided in the samples/admin_scripts directory under your installation directory.

The *wlmtiersdefault.db2* and *wlmtierstimerons.db2* scripts create three service subclasses under a common superclass with successively lower resource settings from high to low along with CPUTIMEINSC thresholds that move or remap activities in response to the consumption of processor time. The scripts differ in how activities are mapped to a service class when they first enter your data server. A third script, *wlmtiersdrop.db2*, drops the WLM objects created by the other two sample tiering scripts.

**wlmtiersdefault.db2**

All activities entering the data server are mapped to a high priority service subclass WLM_SHORT without differentiation between types of work. Activities will complete in the high priority service subclass whilst the highest priority is assigned to them, unless they exceed the maximum amount of processor time they are allowed to consume. Activities that consume too much processor time are first remapped by a threshold REMAP ACTIVITY action to a medium priority service subclass WLM_MEDIUM and then, if they still do not complete without exceeding the allotted processor time in that subclass, remapped to a low priority service subclass WLM_LONG, where they continue to be processed until they complete. Activities that cannot be remapped by a CPUTIMEINSC threshold are mapped directly to service subclass WLM_MEDIUM where they will remain.

**wlmtierstimerons.db2**

DML activities entering the data server are evaluated according to their estimated cost and mapped to one of the three service subclass. DML activities estimated to be short are mapped to a high priority service subclass WLM_SHORT, DML activities estimated to be of medium length are mapped to a WLM_MEDIUM service subclass that receives medium priority, and long DML activities are mapped to a WLM_LONG service subclass that receives the lowest priority. Non-DML activities enter the highest priority service subclass. As they are processed, activities that consume more processor time than assigned to a service subclass are successively remapped by a threshold REMAP ACTIVITY action to the next lowest priority service subclass, until they are remapped to the lowest priority service subclass where they continue to be processed until they complete. Activities that cannot be remapped by a CPUTIMEINSC threshold are mapped directly to service subclass WLM_MEDIUM where they will remain.

**wlmtiersdrop.db2**

This script drops all DB2 workload manager service classes, thresholds, workloads, work class sets and work action sets that are created by the scripts *wlmtiersdefault.db2* and *wlmtierstimerons.db2*.

By default, the *wlmtiersdefault.db2* and *wlmtierstimerons.db2* scripts use the following service class and threshold definitions:
Table 39. Service classes with CPU shares and prefetch priority settings created by the scripts

<table>
<thead>
<tr>
<th>Service class</th>
<th>CPU shares</th>
<th>Prefetch priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_SHORT (high priority)</td>
<td>6000</td>
<td>High</td>
</tr>
<tr>
<td>WLM_MEDIUM (medium priority)</td>
<td>3000</td>
<td>Medium</td>
</tr>
<tr>
<td>WLM_LONG (low priority)</td>
<td>1000</td>
<td>Low</td>
</tr>
<tr>
<td>Default system class</td>
<td>Default</td>
<td>High</td>
</tr>
<tr>
<td>Default maintenance class</td>
<td>Default</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 40. Thresholds created by the scripts

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Maximum amount of processor time that can be used in the service class before remapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_TIERS_REMAP_SHORT_TO_MEDIUM</td>
<td>30 seconds</td>
</tr>
<tr>
<td>WLM_TIERS_REMAP_MEDIUM_TO_LONG</td>
<td>30 seconds</td>
</tr>
</tbody>
</table>

The wlmtiersdefault.db2 sample script creates the following work action set and work class set, which is used to map activities that cannot be remapped by the CPUTIMEINSC threshold directly to the WLM_MEDIUM service subclass. These activities will remain in the WLM_MEDIUM service subclass for the duration of their execution.

Table 41. Work class set created by the wlmtiersdefault.db2 sample script

<table>
<thead>
<tr>
<th>Work class</th>
<th>Work action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_DML_WC</td>
<td>For DML activities, mapped to service class WLM_SHORT initially. These activities can be remapped by a CPUTIMEINSC threshold.</td>
</tr>
<tr>
<td>WLM_CALL_WC</td>
<td>For CALL activities, mapped to service class WLM_SHORT initially. These activities can be remapped by a CPUTIMEINSC threshold.</td>
</tr>
<tr>
<td>WLM_OTHER_WC</td>
<td>For activities that cannot be remapped by a CPUTIMEINSC threshold, mapped to service class WLM_MEDIUM. These activities will remain in the WLM_MEDIUM service subclass.</td>
</tr>
</tbody>
</table>

The wlmtierstimerons.db2 sample script also creates the following work action set and work class set, which is used to map activities according to their estimated cost:

Table 42. Work class set created by the wlmtierstimerons.db2 sample script

<table>
<thead>
<tr>
<th>Work class</th>
<th>Estimated cost range in timerons and work action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_SHORT_DML_WC</td>
<td>For DML activities with an estimated cost of 0 to 999 timerons, mapped to service class WLM_SHORT initially. These activities may get remapped by a CPUTIMEINSC threshold.</td>
</tr>
</tbody>
</table>
### Table 42. Work class set created by the `wlmtierstimerons.db2` sample script (continued)

<table>
<thead>
<tr>
<th>Work class</th>
<th>Estimated cost range in timerons and work action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_MEDIUM_DML_WC</td>
<td>For DML activities with an estimated cost of 1000 to 99,999 timerons, mapped to service class WLM_MEDIUM initially. These activities may get remapped by a CPUTIMEINSC threshold.</td>
</tr>
<tr>
<td>WLM_LONG_DML_WC</td>
<td>For DML activities with an estimated cost of 100,000 to infinity timerons, mapped to service class WLM_LONG.</td>
</tr>
<tr>
<td>WLM_CALL_WC</td>
<td>For CALL activities, mapped to service class WLM_SHORT initially. These activities can be remapped by a CPUTIMEINSC threshold.</td>
</tr>
<tr>
<td>WLM_OTHER_WC</td>
<td>For activities that cannot be remapped, mapped to service class WLM_MEDIUM</td>
</tr>
</tbody>
</table>

### Modifying the scripts for your environment

When you modify the sample scripts to adapt them to your environment, the most important setting to consider is the maximum amount of processor time that can be used in each service class. How much processor time you permit activities to consume in each service subclass depends largely on your particular environment. To find the best values, you need to monitor how activities are being processed on your data server. By default, both the `wlmtiersdefault.db2` and `wlmtierstimerons.db2` scripts will log event monitor records to the threshold violations event monitor, if one is active, with the option to turn on and enable the activity event monitor and to collect activity data (at the cost of incurring additional overhead). For `wlmtiersdefault.db2`, if the maximum amount of processor time that can be used in each service class is set too high, most activities will always start and finish in the high priority class regardless of how much actual processor time each requires. If the maximum amount of processor time is set too low, no activity will finish in the high priority service class and every activity will end up being remapped to the medium or low priority service class regardless of business priority. In either case, the script will not benefit overall throughput on your data server and activities are not treated according to their business priority effectively. The same issue is true to a lesser extent for `wlmtierstimerons.db2` where activities are differentiated initially by being mapped to service subclasses according to estimated cost. If the maximum amount of processor time that can be used in each service class is set incorrectly, activities will fail to be remapped to a more appropriate service subclass if they consume too much processor time, or are remapped too quickly despite having higher business priority.

For more information about the specific DB2 workload manager objects created by the scripts and about how to run them, refer to the scripts.

### Sample scenarios

Two examples have been included in the documentation that show you how you can adapt the sample tiering scripts on your data server to make use of priority aging.
Scenario: Controlling resource intensive business intelligence reports with priority aging

The following scenario shows how you can configure your data server to dynamically lower the priority of expensive business intelligence reports that cannot be identified before execution starts in order to maintain system performance for other queries.

The problem: There is a business intelligence report which any end user can run and which is very expensive. Anytime the report runs, it compromises the performance of the system. The front end tool used to generate the report does not set any client information that could be used to identify the report in advance which would permit you to map it to a low priority service class using a workload.

The solution: You can use the `wlmtiersdefault.db2` sample tiering scripts to configure your data server with a tiered configuration that dynamically lowers, or ages, the priority of processor intensive activities during their lifetime in order to prevent compromising data server performance for all other users. After a workload initially maps all work to a high priority service subclass, the expensive reports are detected by the CPUTIMEINSC in-service-class threshold based on the amount of processor time consumed. If an activity violates the CPUTIMEINSC threshold by using the maximum amount of allowed processor time, a REMAP ACTIVITY moves the activity to a lower priority service subclass. The activity can be remapped in response to processor time consumption again until it executes in the lowest priority service subclass where it will continue until it completes or you intervene manually. Other activities which do not exceed the thresholds continue to run in the high priority service subclass, where they receive more system resources.

An event monitor record is logged every time an activity is remapped, if you created a threshold violations event monitor. If you want to collect additional information about remapped activities to investigate further, you can add the COLLECT ACTIVITY DATA clause to the ALTER THRESHOLD statement in the `wlmtiersdefault.db2` script. Simply rerun the script for the change to take effect.

After running the workload for a period of time, you can use the `WLM_GET_SERVICE_SUBCLASS_STATS` table function to see how many activities were remapped between the service subclasses:

```sql
SELECT substr(service_superclass_name,1,21) AS superclass,
       substr(service_subclass_name,1,21) AS subclass,
       substr(char(coord_act_completed_total),1,10) AS completed,
       substr(char(coord_act_remapped_in),1,10) AS remapped_in,
       substr(char(coord_act_remapped_out),1,10) AS remapped_out,
       substr(char(last_reset),1,19) AS last_reset
FROM table( WLM_GET_SERVICE_SUBCLASS_STATS(
           CAST(NULL AS VARCHAR(128)),
           CAST(NULL AS VARCHAR(128)),
           -2 ) ) AS TF_subcls_stats0

SELECT SUBSTR(WORKLOAD_NAME,1,19) AS WL_NAME,
       COORD_ACT_LIFETIME_AVG,
       COORD_ACT_LIFETIME_STDDEV
FROM TABLE(WLM_GET_WORKLOAD_STATS(
           CAST(NULL AS VARCHAR(128)), -2))
AS WLSTATS
ORDER BY WL_NAME
```

<table>
<thead>
<tr>
<th>SUPERCLASS</th>
<th>SUBCLASS</th>
<th>COMPLETED</th>
<th>REMAPPED IN</th>
<th>REMAPPED_OUT</th>
<th>LAST_RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTSYSTEMCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2008-10-06-20.53.47</td>
</tr>
<tr>
<td>SYSDEFAULTMAINTENANCE</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2008-10-06-20.53.47</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2008-10-06-20.53.47</td>
</tr>
<tr>
<td>WLM_TIERS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2008-10-06-20.53.47</td>
</tr>
<tr>
<td>WLM_TIERS</td>
<td>WLM_SHORT</td>
<td>999</td>
<td>0</td>
<td>35</td>
<td>2008-10-06-20.53.47</td>
</tr>
</tbody>
</table>

Chapter 3. Activities management 153
If you notice that no or only very few activities are being remapped to the lower priority service subclasses, decrease the CPUTIMEINSC threshold value and the check interval used by the ALTER THRESHOLD statements in the script to improve the mapping of activities across service class tiers according to business priority. If most or almost all activities are being remapped to the lower priority service subclasses, increase the CPUTIMEINSC threshold value and the check interval for the ALTER THRESHOLD statements to permit more activities to complete with higher priority. After your changes are complete, rerun the wlmtiersdefault.db2 script to make them effective.

**Scenario: Remapping incorrectly mapped queries through priority aging**

The following scenario shows how you can configure your data server to dynamically remap, or age the priority of, activities that are consuming more processor time than originally estimated in order to maintain system performance for other queries.

The problem: You may have mapped expensive activities based on estimated SQL cost to a lower priority service subclass so that these activities do not impact the performance of less expensive, shorter activities. Such a mapping can be accomplished by defining a work action set at the service superclass level. However, if the estimated SQL cost is incorrect because of statistics that are out of date, for example, an expensive activity might be mapped incorrectly to a high priority service subclass where it begins to consume an excessive amount of resources, at the cost of all other high priority activities.

The solution: You can use the wlmtierstimerons.db2 sample tiering script to configure your data server with a tiered configuration that evaluates incoming activities according to their estimated cost and maps them to one of three service subclasses, each with different shares of CPU. If an activity consumes too much processor time, your data server dynamically lowers the priority of the activity during its lifetime by remapping it between performance tiers. This dynamic process of remapping activities to lower their priority is also referred to as priority aging.

After an activity has been mapped to its initial service class and begins executing, the CPUTIMEINSC in-service-class threshold is used by the script to control the amount of processor time an activity can consume. If the activity violates the threshold by using the maximum amount of allowed processor time, a REMAP ACTIVITY action is triggered which moves the activity to a service subclass with less CPU resources allocated. The activity can be remapped in response to processor time consumption until it executes the lowest priority service subclass where it will continue until it completes or you intervene manually.

An event monitor record is logged every time an activity is remapped. If you want to collect additional information about remapped activities to investigate further, you can add the COLLECT ACTIVITY DATA clause to the ALTER THRESHOLD statement in the wlmtiersdefault.db2 script. Simply rerun the script for the change to take effect.
After running the workload for a period of time, you can use the
WLM_GET_SERVICE_SUBCLASS_STATS table function to see how many activities
were remapped between the service subclasses:

```sql
SELECT substr(service_superclass_name,1,21) AS superclass,
       substr(service_subclass_name,1,21) AS subclass,
       substr(char(coord_act_completed_total),1,10) AS completed,
       substr(char(act_remapped_in),1,10) AS remapped_in,
       substr(char(act_remapped_out),1,10) AS remapped_out,
       substr(char(last_reset),1,19) AS last_reset
FROM table( WLM_GET_SERVICE_SUBCLASS_STATS(
              CAST(NULL AS VARCHAR(128)),
              CAST(NULL AS VARCHAR(128)),
              -2 )
        ) AS TF_subcls_stats
```

```sql
SELECT SUBSTR(WORKLOAD_NAME,1,19) AS WL_NAME,
       COORD_ACT_LIFETIME_AVG,
       COORD_ACT_LIFETIME_STDDEV
FROM TABLE(WLM_GET_WORKLOAD_STATS
            (CAST(NULL AS VARCHAR(128)), -2))
AS WLSTATS
ORDER BY WL_NAME
```

<table>
<thead>
<tr>
<th>SUPERCLASS</th>
<th>SUBCLASS</th>
<th>COMPLETED</th>
<th>REMAPPED_IN</th>
<th>REMAPPED_OUT</th>
<th>LAST_RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTSYSTEMCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2008-10-06-20.59.27</td>
</tr>
<tr>
<td>SYSDEFAULTMAINTENANCE</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2008-10-06-20.59.27</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2008-10-06-20.59.27</td>
</tr>
<tr>
<td>WLM_TIERS</td>
<td>WLM_SHORT</td>
<td>651</td>
<td>0</td>
<td>5</td>
<td>2008-10-06-20.59.27</td>
</tr>
<tr>
<td>WLM_TIERS</td>
<td>WLM_MEDIUM</td>
<td>36</td>
<td>5</td>
<td>7</td>
<td>2008-10-06-20.59.27</td>
</tr>
<tr>
<td>WLM_TIERS</td>
<td>WLM_LONG</td>
<td>16</td>
<td>7</td>
<td>0</td>
<td>2008-10-06-20.59.27</td>
</tr>
</tbody>
</table>

7 record(s) selected.

For this scenario, you should see relatively few activities being remapped between
service subclasses, because activities should almost always be mapped to the
appropriate service subclass initially, based on estimated cost. If you notice that
activities typically are being completed only in the WLM_SHORT or the
WLM_LONG service class, you can adjust the estimated cost values used by the
ALTER WORK CLASS SET statement in the script to improve the mapping of
activities across service class tiers, so that shorter activities are mapped to the
WLM_SHORT_DML_WC work class and longer activities are mapped to the
WLM_MEDIUM_DML_WC or the WLM_LONG_DML_WC work class. If you
notice that most of the activities are being remapped, you can increase the
threshold values used in the ALTER THRESHOLD statements to improve the
initial mapping of activities to service subclasses. After your changes are complete,
rerun the wlmtierstimerons.db2 script to make them effective.

**Remapping activities between service subclasses**

You enable remapping by specifying a REMAP ACTIVITY action with CREATE
and ALTER THRESHOLD statements. The remap action, when it is triggered by a
threshold violation, moves an activity from one service subclass to another service
subclass under the same superclass.

**Before you begin**

In order to be able to remap to another service subclass, the target service subclass
must exist under the same service superclass as the original service subclass of the
activity. Either the target or original service subclass can be the default subclass of
the superclass. The REMAP ACTIVITY action cannot be applied to service
subclasses under the default system class, default maintenance class or default user
class.
About this task

The REMAP ACTIVITY action will move an activity to a different service subclass within the same service superclass. Remapping is available with any of the in-service-class thresholds such as CPUTIMEINSC and SQLROWSREADINSC. You use this dynamic process of remapping activities to lower their priority over time, which is also known as priority aging. Lowering the priority of some activities over time can free up system resources, which can then be applied to other activities of higher business importance.

You can use the REMAP ACTIVITY action with the DATATAGINSC in-service-class threshold which maps an activity to a different DB2 service subclass based on the data that is estimated to be accessed.

Agents working for the activity will periodically check if a threshold has been violated on each member, without coordination between members. When any one agent detects an in-service-class threshold violation on a member, this agent triggers the REMAP ACTIVITY action for the activity on the member and then remaps itself to the target service subclass, after which the activity is considered remapped. All other agents working for the activity on the same member will remap to the target service subclass when they detect that the activity has been remapped.

Restrictions

The target service subclass cannot be the same as the original service subclass; you must remap to a different service subclass first before remapping to the original one.

If an activity is remapped to a disabled service subclass, the activity is treated as if it has been rejected by the disabled subclass and an error message 4714 is returned to the client.

Procedure

1. Identify those activities which you want to control through priority aging. A tiered setup consists of service subclasses under the same service superclass that have in-service-class thresholds with REMAP ACTIVITY threshold actions defined on them. You can refer to the basic priority aging example and to the sample scenarios as a starting point:
   a. “Priority aging of ongoing work” on page 145
   b. “Scenario: Controlling resource intensive business intelligence reports with priority aging” on page 153
   c. “Scenario: Remapping incorrectly mapped queries through priority aging” on page 154

2. Select the different service subclasses that activities will be mapped to. This includes both the the service subclass an activity is originally mapped to when activity execution starts, and any other service subclass or classes the activity will be remapped to during its lifetime. For more information on service classes, including on how to create them, see “Resource assignment with service classes” on page 67.

3. Create or alter your thresholds for controlling activities. For more information on thresholds, see “Priority aging of ongoing work” on page 145.
   a. Define your in-service-class thresholds so that they include a REMAP ACTIVITY action, which is triggered when the threshold is violated. Note
that an in-service-class threshold applies to and is affected by an activity only while the activity is mapped to the associated service subclass; affected counters and timers are reset after remapping. Consider if a threshold violation record should be logged each time an activity remaps. These records provide information about which service classes an activity spent time executing in, which you can use for performance analysis. Note that logging threshold violation records can begin consuming substantial amounts of disk space, if remapping of activities between service subclasses is a common occurrence.

b. You can also define any thresholds that you want to apply to the lifetime of the activity, but note that only thresholds from the first service subclass that the activity is originally mapped to continue to apply throughout the lifetime of the activity. If you also define any thresholds on any of the service subclasses that an activity is later remapped to, they do not apply.

4. Commit your changes. When you commit your changes, your thresholds are added to the SYSCAT.THRESHOLDS view.

5. Allow your data server to execute the activities you are targeting with your in-service-class thresholds and monitor their progress during their lifetime. Activities will stay in the their original service subclass during execution as long as they do not violate an in-service-class threshold. As in-service-class thresholds are violated during activity execution, activities will trigger a REMAP ACTIVITY action that dynamically remaps them to different service subclasses. Once remapped, the activities continue execution and are now controlled by the resource constraints you have placed on the target service subclass.

6. If necessary, refine your approach to priority aging to reach your stated performance goals.

Example

The following example creates a simple three-tiered setup that lowers, or ages, the priority of ongoing activity over time. Three service subclasses under a single superclass A provide the execution environment in which all queries must run. Assume that the default user workload maps incoming queries to service subclass A1, which is a high-priority subclass intended to permit shorter running queries to execute quickly. A medium-priority service subclass A2 is intended to permit longer running queries to execute, although with more stringent resource controls. Service subclass A3 provides containment for any very large queries that take an excessive amount of processor time to complete.

Three thresholds provide control over the ongoing resource consumption of queries. A query is permitted to execute in the high-priority service subclass A1 only as long as it requires less than one minute of processor time to complete. After a minute of processor time has been consumed, threshold T1 automatically remaps the activity to subclass A2, where it can continue executing as long as it consumes less than 10 minutes of processor time. If the query still has not completed after 10 minutes of consuming processor time, threshold T2 remaps the activity to the lowest priority service subclass, A3. Queries in subclass A3 are permitted to continue indefinitely, although an event monitor record is logged and activity data with details is collected when the processor time used exceeds 1 hour.

CREATE SERVICE CLASS A
CREATE SERVICE CLASS A1 UNDER A
CREATE SERVICE CLASS A2 UNDER A
CREATE SERVICE CLASS A3 UNDER A

CREATE THRESHOLD T1 FOR SERVICE CLASS A1 UNDER A
Workload management dispatcher overview

The DB2 workload management dispatcher is a built-in DB2 technology by which you can specifically allocate CPU resources to work that is being executed on a database server. CPU resource entitlements can be controlled by using CPU shares and CPU limit attributes on DB2 service classes.

Overview

Particularly useful under workload conditions that strain the system CPU resources, the dispatcher can effectively manage service class CPU resource entitlements with attributes such as the uncapped (almost unrestricted) soft CPU shares, and the two capped (limited) attributes: hard CPU shares and CPU limits. You can use the uncapped soft CPU shares to give high-priority-work service classes any unused CPU resources. In addition, you can enforce controls on the CPU resource entitlements by using capped hard CPU shares or CPU limits assigned to low-priority-work service classes, thereby limiting their impact on high-priority work. Under conditions where the system CPU resources are less strained, you can use one or both of the capped attributes, without the need to assign uncapped attributes, to effectively control the CPU resource entitlements between service classes running in a typical, underrun-CPU environment.

The workload management dispatcher infrastructure operates at the instance level in the DB2 database manager. The dispatcher operates by limiting the number of executing agents to be dispatched to the operating system (OS) and how long each agent is allowed to execute, at any given time. The number of executing agents that can be dispatched at the same time is referred to as the dispatch concurrency level.

The DB2 workload management dispatcher has the following benefits:

- Easy to implement, requiring less time and effort than implementing an OS WLM such as AIX WLM or Linux WLM.
- Supports flexible CPU allocation throughout the normal daily ebb and flow of system use. This flexibility is accomplished through the provisioning of both permanent allocations that are enforced all the time (hard CPU shares and CPU limits), or dynamic allocations that are enforced only when demand exceeds capacity (soft CPU shares).
- Self-contained within the DB2 database manager such that setting CPU resource entitlements provides you with workload control that is effective across all platforms due to its independence from an OS WLM such as AIX WLM or Linux WLM.
- You can continue to use the OS WLM products as a workload control mechanism, but it is not necessary if the extra implementation complexity (for example, setting up AIX WLM on each member) or if the organizational strife
(for example, system administrator reluctance to implement or permit use of OS WLM) is a barrier. Alternatively, you can use the OS WLM products for monitoring purposes while relying on the DB2 workload management dispatcher for workload control.

More information

This section provides information about essential administration concepts, tasks, and usage scenarios for the following areas:

- DB2 workload management dispatcher
- Hard CPU shares
- Soft CPU shares
- CPU limit
- Minimum CPU utilization level to have service class considered as active
- Dispatch concurrency level
- Monitoring and tuning workload management dispatcher performance

Workload management dispatcher

The workload management dispatcher manages CPU resource entitlements for workloads running within the DB2 database manager. Details about its features and functionality are provided here.

Introduction

You can use the built-in DB2 workload management dispatcher to manage DB2 workloads by configuring CPU resource entitlements without the need for third-party workload management software. The dispatcher can control DB2-specific CPU resource consumption by means of CPU shares-based settings and CPU limit settings. Soft CPU shares represent an uncapped resource allocation and can be used to allow high-priority work to consume more than its allocated share when there are unused CPU resources. Hard CPU shares and CPU limits represent a capped CPU allocation that can be used to prevent low-priority work from impacting the execution of high-priority work. CPU limits can be used to strictly control the CPU that is consumed by a particular workload in order to provide isolation for other work running on the system, whereas hard CPU shares provide the additional flexibility of allowing low-priority work to consume unused CPU resources in the absence of a high-priority workload. This scenario might occur during off-peak business hours.

The sections that follow describe the features and functionality of the workload management dispatcher in more detail. A scenarios section helps to illustrate the workload management dispatcher features and functionality with usage examples.

Features and functionality

The workload management dispatcher infrastructure operates at the instance level in the DB2 database manager. However, at the database level, you specify the CPU resource entitlements using the CPU shares and CPU limit attributes that can be placed on a service class through the CREATE SERVICE CLASS and ALTER SERVICE CLASS statements.

As mentioned, the hard and soft CPU shares attributes have different conditions under which unused CPU resources, relinquished by a service class or a number of service classes, can be claimed by the remaining competing service classes.
However, both hard and soft CPU shares attributes have the same behavior when it comes to giving back those claimed CPU resources. For example, when a service class relinquishes some or all of its full share of CPU resources, those service classes that claimed the unused CPU resources must give back the CPU resources to the service class trying to retrieve its full share when it begins to increase its demand for CPU resources. This will lead to the full restoration of the user-specified CPU resource entitlement to the retrieving service class if the demand is present. Every second, a DB2 database manager scheduler thread evaluates the CPU resource utilization by work in progress and make adjustments to ensure, over time, that the dispatcher is providing the configured CPU shares and CPU limit allocations.

The workload management dispatcher operates by limiting both the number of executing agents simultaneously dispatched to the operating system (OS) and how long each agent is allowed to execute. The number of executing agents that can be dispatched at the same time is referred to as the dispatch concurrency level which you can set using the `wlm_disp_concur` database manager configuration parameter.

By default, the workload management dispatcher is not enabled. To control DB2 database manager-specific CPU resource entitlements, you must first enable the workload management dispatcher. To enable the workload management dispatcher, you must set the `wlm_dispatcher` database manager configuration parameter to `YES` (by default, this configuration parameter is set to `NO`). For complete details about how to enable the workload management dispatcher, see: “Enabling workload management dispatcher” on page 177.

When the workload management dispatcher is enabled, all work running in the user and maintenance service classes within the DB2 database manager is under the control of the dispatcher. Work running in the system service class cannot be configured for CPU resource control because critical DB2 subsystems that run in this service class are given maximum priority and are not subject to workload management dispatcher control. By default, once turned on, the dispatcher can manage CPU resources only by way of CPU limit settings. To allow the dispatcher to manage CPU resources using both CPU shares and CPU limits, you must set the `wlm_disp_cpu_shares` database manager configuration parameter to `YES`.

Before or after enabling the workload management dispatcher for the first time, you will want to monitor your workloads to determine the relative CPU resources that they consume. Based on the relative priority of the workloads, this information can help in formulating your decisions concerning the creation of service classes to which the work can be assigned, the type of CPU shares (hard or soft) to assign to particular service classes, the relative quantity of CPU shares to assign to each service class, and whether to use CPU limits.

Another consideration that gives you maximum flexibility to control the behavior of the DB2 database manager is the option to set the minimum percentage of CPU resource utilization for service classes to be considered active. After setting the `wlm_disp_min_util` database manager configuration parameter, service classes utilizing CPU resources equal to or greater than that minimum percentage are considered to be active on the host or logical partition (LPAR) and the CPU shares of the active service classes are factored into CPU resource entitlement calculations.

Through the workload management dispatcher, you can control the allocation of CPU resources between service classes on your system via the service class CPU shares attribute. The shares attribute represents the relative entitlement of CPU time that each service class receives compared to other service classes on the
system. By allocating more CPU shares to one service class and less to another, you can control the amount of CPU resources allocated to each service class and provide better quality of service to one service class over another.

After having made the decision that the earlier enabled workload management dispatcher can best manage your strained CPU resources by using CPU shares along with CPU limits, you must enable CPU shares by setting the `wlm_disp_cpu_shares` database manager configuration parameter to `YES`. The default setting for this parameter is `NO`. You can set and adjust CPU shares and CPU limits by using the `CREATE SERVICE CLASS` and `ALTER SERVICE CLASS` statements.

Based on the number of CPU shares assigned to a service class, the workload management dispatcher calculates the proportion of the CPU resources that each service class is entitled to use. To determine the proportion of the CPU resources to which each service superclass is entitled, you can use the following formula to convert the number of CPU shares of a particular service superclass into a percentage of CPU resources allocated by the workload management dispatcher:

\[ \% \text{CPU(superclass)} = \frac{\text{Number of superclass shares}}{\text{Total number of shares of all active superclasses}} \times 100 \]

To determine the proportion of the CPU resources to which each service subclass is entitled, you can use the following formula to convert the number of CPU shares of a particular service subclass into a percentage of CPU resources allocated by the workload management dispatcher:

\[ \% \text{CPU(subclass)} = \% \text{CPU(superclass)} \times \frac{\text{Number of subclass shares}}{\text{Total number of shares of all active subclasses in the superclass}} \]

**Note:** The total number of CPU shares (both hard and soft) of all active superclasses are counted across all databases and all members on the host or LPAR.

For illustrative examples of CPU proportion calculations, see the “Usage scenarios” section.

**Summary of CPU resource scheduling capabilities**

The following table summarizes the features of the different CPU resource control attributes available for DB2 service classes managed by the workload management dispatcher.
Table 43. Summary of workload management dispatcher CPU resource scheduling capabilities

<table>
<thead>
<tr>
<th>CPU resource control attribute</th>
<th>Usage of idle CPU resources in the same context</th>
<th>CPU resource entitlement limits are relative to other active service classes in the same context</th>
<th>CPU resource entitlement limits are fixed based on overall system CPU capacity</th>
<th>Control of CPU consumption at high CPU resource utilization</th>
<th>Control of CPU consumption at low CPU resource utilization</th>
<th>Usage notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soft shares</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Ideal for high-priority work to exploit any spare CPU resources available on the host or LPAR to maximize workload progress</td>
</tr>
<tr>
<td>Hard shares</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Ideal for ensuring work does not interfere with higher priority work which might be running on the host or LPAR in various amounts and intensity</td>
</tr>
<tr>
<td>CPU limits</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Ideal for sandboxing and enforcing consistent fixed CPU resource allocations between service classes, or limiting service class CPU resource consumption regardless of other work on the host or LPAR</td>
</tr>
</tbody>
</table>

**Effects of work outside of workload management dispatcher control**

A workload management dispatcher exists for each DB2 instance. It can only directly detect and manage DB2 work within its own instance. Any other work running on the same host or LPAR is not under the direct control of the dispatcher. The random nature of the CPU resource demands of work running outside of the
DB2 database manager means that while the CPU limits and the relative shares of CPU resources among work executing within DB2 can be enforced, at any given instant of time, the work managed by the dispatcher still needs to compete with work running outside its own DB2 instance and outside the dispatcher’s direct control. Work that is neither detected nor managed by the workload management dispatcher includes the following:

- Work performed by applications or middleware products that perform a portion of their work outside of the DB2 database manager
- Work performed by entities executing in the DB2 system service class
- Work performed by other DB2 instances
- Non-DB2 work performed in fenced mode processes such as fenced stored procedures
- Non-DB2 work performed in trusted routines. Trusted procedures and user-defined functions (UDFs) are accounted for in dispatcher CPU resource scheduling, but the dispatcher is not able to limit CPU utilization when a trusted routine executes in user code for long periods of time.

Let’s consider that you have a host or LPAR that runs only the DB2 database manager. You have created a database with service classes A and B and assigned a 50% share of the CPU resources to each service class. Assuming the overall host or LPAR makes full use of the CPU resources at all times, your measurements of total CPU utilization (%) over time looks similar to that shown in Figure 23 on page 164 panel A.
Now, let's consider what happens when fenced mode processes (FMPs) are executed on your host or LPAR. These processes are not under the control of the workload management dispatcher, so the amount of CPU resources they use varies randomly. However, out of the remaining CPU resources that the dispatcher can manage, service classes A and B each continue to get an equal 50% split of the leftover CPU resources, as shown in panel B.

**Interactions between dispatcher and other service class settings**

The following table summarizes how the workload management dispatcher settings on a service class (hard and soft CPU shares, and CPU limit) interact with other settings on the same service class.

<table>
<thead>
<tr>
<th>Dispatcher service class setting</th>
<th>Other service class setting</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard and soft CPU shares, and CPU limit</td>
<td>Outbound correlator (OS WLM integration)</td>
<td>Hard and soft CPU shares, CPU limit, and OS WLM can all be active at the same time.</td>
</tr>
</tbody>
</table>
**Table 44. Interactions between dispatcher and other service class settings (continued)**

<table>
<thead>
<tr>
<th>Dispatcher service class setting</th>
<th>Other service class setting</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard and soft CPU shares, and CPU limit</td>
<td>Buffer pool priority</td>
<td>Hard and soft CPU shares, CPU limit, and buffer pool priority can all be active at the same time.</td>
</tr>
<tr>
<td>Hard and soft CPU shares, and CPU limit</td>
<td>Prefetch priority</td>
<td>Hard and soft CPU shares, CPU limit, and prefetch priority can all be active at the same time.</td>
</tr>
</tbody>
</table>

**Interaction between dispatcher settings and thresholds with a REMAP action**

Some workload management thresholds support a REMAP ACTIVITY action to move executing work from one service subclass to another within the same service superclass. When work moves to a service subclass with a different CPU share or CPU limit than the service subclass it moved from, the CPU share and CPU limit settings of the service subclass to which the work moves are applied.

**Comparison between workload management dispatcher and workload management concurrency thresholds**

Although you can control workload scheduling and resource consumption using both the workload management dispatcher and concurrency thresholds, knowing the differences in their degree of workload control and benefits will help you to decide when to apply one or both in your environment. The following table compares their abilities to control CPU resources.

**Table 45. Comparison of CPU resource control abilities between workload management dispatcher and workload management concurrency thresholds**

<table>
<thead>
<tr>
<th>Workload management dispatcher</th>
<th>DB2 workload management concurrency thresholds</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can directly control CPU allocations within the DB2 database manager after an activity begins consuming CPU resources.</td>
<td>You can indirectly control both the consumption of CPU and other resources, such as memory, by limiting the number of connections or activities that are allowed to execute at the same time.</td>
</tr>
<tr>
<td>You can configure relative allocations of soft and hard shares of CPU resources to service classes, in addition to absolute limits of CPU consumption by a single service class.</td>
<td>You can impose an absolute limit on the concurrency of executing activities or open connections for a service class, but you cannot specify any kind of relative CPU allocations.</td>
</tr>
</tbody>
</table>

**Comparison between workload management dispatcher and OS WLMs (AIX WLM and Linux WLM)**

The workload management dispatcher can be used together with an OS WLM (AIX WLM or Linux WLM) in certain situations to provide even greater control. The following are some examples of circumstances in which both the workload management dispatcher and an OS WLM can work together:

- OS WLMs often have OS-level monitoring that complements the workload management dispatcher monitoring capabilities.
OS WLMs can manage all applications on the system, including the DBMS. The workload management dispatcher can provide DB2-specific dispatching, while the OS WLM manages all the other applications on the system.

Do not attempt to control CPU resource allocations to individual service classes with both the OS WLM and the workload management dispatcher at the same time. You can integrate an OS WLM at the service class level for monitoring purposes when the workload management dispatcher is enabled. You can also integrate an OS WLM to control external applications, or the overall CPU resources allocated to a DB2 instance, or both.

Table 46. Comparison of abilities between workload management dispatcher and OS WLM

<table>
<thead>
<tr>
<th>Workload management dispatcher</th>
<th>OS WLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A single solution that works across all platforms.</td>
<td>An OS WLM is specific to each OS. Each has its own user interface and its own limitations.</td>
</tr>
<tr>
<td>You can control CPU allocations within the DB2 database manager without the need for any OS-specific integration and/or set up - a DBA can manage these settings without requiring root access to the system.</td>
<td>An OS WLM often requires root privilege for configuration.</td>
</tr>
<tr>
<td>You can leverage database-specific knowledge when making CPU resource scheduling decisions that would otherwise not be possible with OS-level facilities.</td>
<td>An OS WLM treats the DB2 database manager as just another application and, therefore, cannot use database-specific knowledge to make CPU resource scheduling decisions.</td>
</tr>
<tr>
<td>In multiple member environments, all members can be configured with one command. However, the result is that you cannot have a different configuration at each member.</td>
<td>In multiple member environments, an OS WLM must be configured at every member. You can have a different configuration at each member.</td>
</tr>
<tr>
<td>Controls the use of CPU resources by only DB2 work running in the user and maintenance service classes within the scope of a single DB2 instance. The dispatcher cannot control the use of CPU resources by applications executing external to the DB2 instance.</td>
<td>An OS WLM can fully control CPU resources on the entire system.</td>
</tr>
<tr>
<td>CPU resource control is accomplished through both soft and hard CPU shares.</td>
<td>OS WLM solutions on AIX and Linux do not currently provide an equivalent of the DB2 hard CPU shares.</td>
</tr>
</tbody>
</table>

Usage scenarios

Scenario 1: CPU entitlement calculations

Figure 24 on page 167 shows the data server configured as illustrated. For illustration purposes to help describe the basic concepts, assume that there is negligible work running in the default user, maintenance, and system service classes. At the start of the scenario, consider that there is enough work executing in the user service superclasses A, B, and C to use the full CPU resource entitlements for those service classes. For the following scenario, there is only one DB2 instance with one database and only one logical member on this host or LPAR.
As shown in Figure 24, the CPU entitlement percentage for each of the service superclasses is calculated as follows:

- Service superclass A has 60%
  \[ \frac{6000}{6000 + 3000 + 1000} \times 100 \]
- Service superclass B has 30%
  \[ \frac{3000}{6000 + 3000 + 1000} \times 100 \]
- Service superclass C has 10%
  \[ \frac{1000}{6000 + 3000 + 1000} \times 100 \]

CPU shares can be assigned at both the service superclass and service subclass levels. If you do not specify CPU shares when creating a service superclass or service subclass, the DB2 database manager assigns a default of 1000 hard CPU shares to such service classes. At the superclass level, the CPU shares value represents how the system CPU resources are divided among DB2 service superclasses. At the subclass level, the CPU shares value represents the division of the CPU resources available to a particular superclass among its active subclasses.

**Scenario 2: Workload management dispatcher operates at instance level; service classes operate at database level**
The workload management dispatcher operates at the instance level, but the configurations (CPU shares and CPU limits) that the dispatcher uses to allocate CPU resources are placed on individual service superclasses and service subclasses within individual databases. When multiple databases exist within the same instance, the sum of the CPU share values within service superclasses on each database determines the relative allocation of the CPU resources that each database receives. For example, there are two databases in the same instance, database A and database B, each database has two service superclasses (SC1 and SC2), as shown in Figure 25 on page 169 panel A.
In this example scenario, we want a 70/30 percentage split of CPU resources in database A which is configured using soft CPU shares of 7000 and 3000 for SC1.

Figure 25. CPU shares allocations to service classes for two databases in one instance
and SC2, respectively, as shown in panel B. In database B, we want a 60/40 percentage split of CPU resources for SC1 and SC2, respectively, as shown in panel C.

Before deciding on how this 60/40 percentage split translates into CPU shares in the two service superclasses for database B, we must decide what are the required relative CPU resource entitlements between databases A and B. The work running in database B is much more important than that running in database A. Therefore, we want database B to have a CPU entitlement of 80% and database A to have 20%.

The sum of the CPU shares on database A is 10,000. These 10,000 CPU shares represent 20% of the total CPU shares across the two databases, we require 50,000 CPU shares in total to represent 100% of the total CPU resources. Therefore, database B must be assigned 80% of those 50,000 CPU shares, resulting in 40,000 CPU shares. To obtain the 60/40 percentage split between the two superclasses of database B, service superclass SC1 receives 60% of the 40,000 total CPU shares assigned in database B, resulting in 24,000 CPU shares. Service superclass SC2 receives 40% of the 40,000 total CPU shares assigned in database B, resulting in 16,000 CPU shares.

The pie chart shows the CPU shares assignments that achieved the percentage CPU resource allocations that we wanted.

**Scenario 3: Favor one workload over another at high CPU utilization and protect favored workload at low CPU utilization**

In this example usage scenario, the aim is to manage the quality of service for multiple workloads while simultaneously maximizing the CPU utilization on the system. However, in addition to experiencing contention for the CPU resources, the workloads in this scenario experience contention for other resources. Allocating unused CPU resources to some workloads when competing workloads, not consuming their full CPU entitlement, are still present, can cause a degradation in performance due to CPU cache contention, I/O contention, or other secondary effects.

In this scenario, there is a need to protect the favored workload from unfavored workloads, despite the possibility that the CPU resources are not fully utilized. The under-utilized CPU resources can result when unfavored workloads are limited and the favored workload is active, but not using its full CPU entitlement. We can accomplish protecting the favored workload by assigning hard CPU shares on the unfavored workloads and assigning soft CPU shares on the favored workload.

Again, let’s use two workloads: ‘Favored’ and ‘Other’. The ‘Favored’ workload is to have 70% of the CPU resources when running at the same time as the ‘Other’ workload and 100% when the ‘Other’ workload is not present. The ‘Other’ workload is to have 30% of the CPU resources when running at the same time as the ‘Favored’ workload and 100% when the ‘Favored’ workload is not present. This is similar to Scenario 3, except that the overall CPU utilization drops below 100% on occasion.

The following steps show how to configure DB2 workload management to achieve the goals listed previously:

1. Create the ‘Favored’ and ‘Other’ service superclasses and assign CPU shares so that the 70/30 CPU entitlement proportion is met. CPU shares can be specified as 7 and 3, 14 and 6, or any other combination that achieves the 70%/30% CPU...
entitlement ratio. In the following example, 7000 soft CPU shares are assigned to the 'Favored' service class and 3000 hard CPU shares are assigned to the 'Other' service class:

```
create service class favored soft cpu shares 7000
create service class other hard cpu shares 3000
```

2. Create the 'Favored' and 'Other' workloads as shown in the following example:

```
create workload favored appl_name('favoredapp') service class favored
create workload other appl_name('otherapp') service class other
```

As a result, the system looks like the following:

**Figure 26. Data server configuration**

For simplicity, assume only a negligible amount of work is running in the service classes other than 'Favored' and 'Other'.

3. Measure whether the desired CPU share goals are being achieved. The system can be in one of several states in terms of its workload:

a. 100% CPU utilization

   1) Work is executing in both the 'Favored' workload and the 'Other' workload
The measured CPU utilization is 70% for the 'Favored' service class and 30% for the 'Other' service class.

2) Work is executing in only the 'Favored' workload
The measured CPU utilization is 100% for the 'Favored' service class and 0% for the 'Other' service class.

3) Work is executing in only the 'Other' workload
The measured CPU utilization is 0% for the 'Favored' service class and 100% for the 'Other' service class.

b. A service class uses less than its CPU entitlement
1) Work is executing in both the 'Favored' workload and the 'Other' workload. The 'Favored' workload is using 50% instead of the full 70% that comes from its CPU entitlement.
   The measured CPU utilization of the 'Favored' workload is 50%. The 'Other' workload continues to use 30% because its CPU shares are hard.
   The total CPU utilization is 80%.

2) Work is executing in both the 'Favored' workload and the 'Other' workload. The 'Other' workload is using 10% instead of the full 30% that comes from its CPU entitlement.
   Due to its CPU shares being soft, the measured CPU utilization of the 'Favored' service class increases from 70% to 90%, claiming the unused 20% relinquished by the 'Other' service class. The CPU utilization of the 'Other' service class is 10%. The total CPU utilization is 100%.

3) Work is executing in only the 'Favored' workload and it uses all of its CPU entitlement
   Due to the absence of work running in the 'Other' service class, the 'Favored' service class can use both its own CPU entitlement of 70% and the unused 30% of CPU resources relinquished by the 'Other' service class. The total CPU utilization is 100%.

4) Work is executing in only the 'Other' workload and it uses all of its CPU entitlement
   Due to the absence of work running in the 'Favored' service class, the 'Other' service class can use both its own CPU entitlement of 30% and the unused 70% of CPU resources relinquished by the 'Favored' service class. The total CPU utilization is 100%.

Scenario 4: Create a sandbox to achieve consistency, even at the expense of some CPU resources becoming idle

In this example usage scenario, a favored workload never consumes more than its fixed proportion of CPU resources, regardless of the availability of unused CPU resources.

Let's use two workloads: 'Favored' and 'Other'. The 'Favored' workload is to have a maximum of 70% of the CPU resources at any time, regardless of the availability of unused CPU resources. The 'Other' workload is to have a maximum of 30% of the CPU resources at any time, regardless of the availability of unused CPU resources.

The following steps show how to configure DB2 workload management to achieve the goals listed previously:
1. Create the 'Favored' and 'Other' service classes and assign each with a CPU limit so that the maximum 70% and 30% of CPU resources is enforced for the 'Favored' and the 'Other' workloads, respectively. In the following example,
CPU limits must be specified as a number between 1 and 100, which represents the maximum percentage of CPU utilization for that service class:

```plaintext
create service class favored cpu limit 7000
create service class other cpu limit 3000
```

2. Create the 'Favored' and 'Other' workloads as shown in the following example:

```plaintext
create workload favored appl_name('favoredapp') service class favored
create workload other appl_name('otherapp') service class other
```

As a result, the system looks like that in Figure 26 on page 171.

For simplicity, assume only a negligible amount of work is running in the service classes other than 'Favored' and 'Other'.

3. Measure whether the desired CPU limits are successfully being enforced. The system can be in one of several states in terms of its workload:

   a. 100% CPU utilization
      1) Work is executing in both the 'Favored' workload and the 'Other' workload
         The measured CPU utilization is 70% for the 'Favored' service class and 30% for the 'Other' service class.
   b. One or more service classes uses less than its CPU limit
      1) Work is executing in both the 'Favored' workload and the 'Other' workload. The 'Favored' workload is using 50% instead of the full 70% that comes from its CPU limit entitlement.
         The measured CPU utilization of the 'Favored' workload is 50%. The 'Other' workload continues to use 30% because that is the maximum CPU limit entitlement. The total CPU utilization is 80%.
      2) Work is executing in both the 'Favored' workload and the 'Other' workload. The 'Other' workload is using 10% instead of the full 30% that comes from its CPU limit entitlement.
         Due to its CPU limit that stipulates a maximum of 70% CPU utilization, the measured CPU utilization of the 'Favored' service class remains at 70%. The unused 20% of CPU resources relinquished by the 'Other' service class remains unclaimed. The CPU utilization of the 'Other' service class is 10%. The total CPU utilization is 80%.
      3) Work is executing in only the 'Favored' workload and it is using the maximum amount of CPU resources as stipulated by its CPU limit
         Due to its CPU limit that stipulates a maximum of 70% CPU utilization, the measured CPU utilization of the 'Favored' service class remains at 70%. The unused 30% of CPU resources relinquished by the 'Other' service class remains unclaimed. The CPU utilization of the 'Other' service class is 0%. The total CPU utilization is 70%.
      4) Work is executing in only the 'Other' workload and it is using the maximum amount of CPU resources as stipulated by its CPU limit
         Due to its CPU limit that stipulates a maximum of 30% CPU utilization, the measured CPU utilization of the 'Other' service class remains at 30%. The unused 70% of CPU resources relinquished by the 'Favored' service class remains unclaimed. The CPU utilization of the 'Favored' service class is 0%. The total CPU utilization is 30%.

Scenario 5: Needing to divide workloads by user and still favor one type of work over another for a given user
In the preceding scenarios, service classes were used to control the allocation of CPU resources between different workloads based on business priorities. In some cases, it might be desirable to both control the share of resources among workloads based on business priorities, and prioritize short-running work over long-running work to improve overall throughput or response times.

In this example usage scenario, a CEO and all of the one thousand other employees access a database through one application named mybizapp. Our aim is to give the CEO alone a full 10% share of the CPU resources and the remaining 90% of the CPU resources is used by the one thousand other users that access the database. To accomplish this goal, create a service superclass for the CEO (ceo_sc). Then, define a workload to map the CEO to her service superclass, as shown in the following examples:

```sql
create service class ceo_sc soft cpu shares 1000
create workload ceo_wl
    applname('mybizapp')
    session_user('ceo')
    service class ceo_sc
```

Create a service superclass for all of the other users (mybizapp_sc). Then, define a workload (mybizapp_wl) to map all of the other users of the mybizapp application to the newly created service superclass mybizapp_sc, as shown in the following example:

```sql
create service class mybizapp_sc hard cpu shares 9000
create workload mybizapp_wl
    applname('mybizapp')
    service class mybizapp_sc
```

Within the workload of the mybizapp application, there are both high-cost and low-cost queries. The business organization has determined that low-cost queries, costing less than 10,000 timerons, are generally higher in priority than high-cost queries, costing more than 10,000 timerons. Therefore, low-cost queries are to be given a greater share of the CPU resources than high-cost queries, specifically by a 2:1 ratio. To accomplish this goal, create two service subclasses (lowcost_ssc and highcost_ssc) within the mybizapp_sc service superclass and assign soft CPU shares in a ratio of 2:1, as shown in the following example:

```sql
create service class lowcost_ssc under mybizapp_sc soft cpu shares 2000
create service class highcost_ssc under mybizapp_sc soft cpu shares 1000
```

The 2:1 ratio of soft CPU shares determines how much of the 90% of the CPU resources assigned to the mybizapp_sc service superclass is allocated to low-cost queries and how much is allocated to high-cost queries.

Create a work class set (splitbycostwcs) and work action set (mybizapp_was) to route low-cost queries to the lowcost_ssc service subclass and high-cost queries to the highcost_ssc service subclass, as shown in the following examples:

```sql
create work class set splitbycostwcs
    (work class lowcost_wc work type dml for timeroncost from 0 to 10000,
     work class highcost_wc work type dml for timeroncost from 10001)
create work action set mybizapp_was for service class mybizapp_sc
    using work class set splitbycostwcs
    (work action maplowcost_wa on work class lowcost_wc map activity
to lowcost_ssc, work action maphighcost_wa on work class highcost_wc
     map activity to highcost_ssc)
```

When work is executing in each of the service classes and consuming its full CPU resource entitlement, the previous configurations result in a division of the CPU
resources, as shown in the following figure:

![Figure 27. CPU resource entitlements](image)

The `ceo_sc` service superclass has 1000 CPU shares out of 10,000, resulting in a CPU resource entitlement of 10%. The `mybizapp_sc` service superclass has 9000 CPU shares out of 10,000, resulting in a CPU resource entitlement of 90%. Within the 90% of the `mybizapp_sc` service superclass CPU resource entitlement, its `lowcostssc` service subclass has 2 shares out of 3, resulting in a CPU resource entitlement of 66.7% of the CPU resources of its service superclass, or 60% of the total CPU resources. The `highcostssc` service subclass has 1 share out of 3, resulting in a CPU resource entitlement of 33.3% of the CPU resources of its superclass, or 30% of the total CPU resources.

**More information**

Complete details are provided about the following subjects:

- Hard CPU shares, see: “Hard CPU shares” on page 178.
- Soft CPU shares, see: “Soft CPU shares” on page 185.
- CPU limit, see: “CPU limit” on page 190.
- Minimum CPU utilization level for a service class to be considered active, see: “Minimum CPU resource utilization for service class to be considered active” on page 205.
- Dispatch concurrency level, see: “Dispatch concurrency level” on page 208.
- Monitoring and tuning workload management dispatcher performance, see: “Monitoring and tuning workload management dispatcher performance” on page 210.

**Workload management dispatcher behavior in multimember environments**

When the workload management dispatcher is enabled on multimember DB2 instances, CPU resource scheduling operates across all of the members on a given host. Note that the workload management dispatcher also supports shared LPAR (micropartition) environments on AIX.

**Multimember database environments**

On multimember DB2 instances, the dispatcher behaves in the following ways:

- The relative quantity of CPU shares are evaluated and CPU resources are allocated across all members in the instance based on the work that is active in that instance on that host or LPAR as a whole. For example, let's consider a two member database with work being executed in two service classes, A and B. Service class A is assigned 3500 soft CPU shares and service class B is assigned 6500 soft CPU shares. The CPU utilization for the entire host or LPAR is 35% for service class A and 65% for service class B despite the possibility that relatively
more work is being executed on one member than the other. The workload management dispatcher looks across both members when making its decisions regarding which service class should be getting more CPU resources at any given moment.

![Diagram of CPU shares and limits across 2 database members]

- The value of the `wlm_disp_concur` database manager configuration parameter applies to each host or LPAR. In multimember databases, the dispatch concurrency level specified by the `wlm_disp_concur` configuration parameter is applied across all of the members.
- For multimember databases, CPU limits are enforced on a per host or LPAR basis - when multiple members are used, each partition is allowed to use all of a specified CPU limit as long as the sum across all members is less than or equal to the CPU limit. If the CPU limit on the single service class A is 35% and there are 2 members, then member 1 can use up to 35% if the sum of the CPU utilization across member 1 and 2 does not exceed 35%.

![Diagram of CPU limit across 2 database members]

**Micropartition (shared LPAR) environments**

The workload management dispatcher supports micropartition environments in AIX. In micropartition environments, the percentage CPU utilization for both controls and monitoring in the workload management dispatcher is computed relative to the CPU resources allocated to the LPAR by the operating system or hypervisor over the most recent dispatcher CPU resource scheduling cycle, using the entitled (guaranteed) CPU resources for the LPAR as a baseline level. In cases when the CPU resource allocated to the LPAR is greater than the entitled CPU...
resources, the dispatcher computes the CPU utilization relative to the greater amount, and when the CPU resource allocation is less than the entitled CPU resources, the dispatcher computes the CPU utilization relative to the entitled CPU resources.

**Note:** The workload management dispatcher behavior is consistent with the way AIX WLM performs its calculations. For dedicated LPARs on AIX, and for all non-AIX environments, the CPU utilization is calculated relative to the full CPU capacity of the physical cores available to the DB2 database manager.

**Enabling workload management dispatcher**

To enable the DB2 workload management dispatcher, set the value of the `wlm_dispatcher` database manager configuration parameter to `YES`. The workload management dispatcher manages CPU resources allocated to DB2 user and maintenance service classes.

**Procedure**

To enable workload management dispatcher by using the command line processor (CLP):

1. Attach to your DB2 instance.
   
   ```sql
   ATTACH TO instance-name
   ```

2. Issue the `UPDATE DATABASE MANAGER CONFIGURATION` (or `UPDATE DBM CFG`) command. Set the value of the `wlm_dispatcher` database manager configuration parameter to `YES`.
   
   ```sql
   UPDATE DBM CFG USING wlm_dispatcher yes
   ```

3. Optional: Detach from your DB2 instance.
   
   ```sql
   DETACH
   ```

**Results**

The workload management dispatcher is enabled. You can now specify CPU limit allocations of CPU resources to DB2 service classes.

**What to do next**

After having enabled the workload management dispatcher to manage CPU resources, you must consider completing the following tasks:

- Create or alter a service class and configure the CPU limit by using the `CREATE SERVICE CLASS` or `ALTER SERVICE CLASS` statement. See: “Setting a CPU limit” on page 204.

- Enable hard and soft CPU shares by configuring the `wlm_disp_cpu_shares` database manager configuration parameter. See step 1 in: “Enabling and setting CPU shares” on page 188.

- Create or alter a service class and configure a hard or soft CPU share by using the `CREATE SERVICE CLASS` or `ALTER SERVICE CLASS` statement. See: “Enabling and setting CPU shares” on page 188.

- Set the minimum CPU utilization percentage for workload management dispatcher to consider service classes to be active by configuring the `wlm_disp_min_util` database manager configuration parameter. See: “Setting minimum CPU resource utilization for service class to be considered active” on page 207.
• Set the dispatch concurrency level by configuring the `wlm_disp_concur` database manager configuration parameter. See: “Setting dispatch concurrency level” on page 209.

**Ensuring maximum DB2 workload management dispatcher scheduling accuracy**

To ensure maximum scheduling accuracy for the workload management dispatcher, the DB2 database manager attempts to assign realtime priorities to the `db2wlmt` timer thread and the `db2wlmtm` scheduling threads. For assignment of such priorities to succeed, certain permissions need to be granted to the DB2 database manager, depending on the operating system.

**Procedure**

Without the ability to assign realtime priorities to the timer and scheduling threads, the DB2 workload management dispatcher is still able to prioritize work, but not as accurately or optimally as it would otherwise. The following tasks, to ensure maximum scheduling accuracy for the workload management dispatcher, are listed according to operating system. Select the operating system that pertains to you and then follow the given instructions.

• On AIX operating systems, the instance owner must have `CAP_NUMA_ATTACH` and `CAP_PROPAGATE` capabilities to set higher priority for the dispatcher threads. To grant these capabilities, logon as `root` and run the following command:

```
chuser capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE
```

• On Solaris 10 or higher, the instance owner must have the `proc_priocntl` privilege to set higher priority for the dispatcher threads. To grant this privilege, logon as `root` and run the following command:

```
usermod -K defaultpriv=basic,proc_priocntl db2user
```

In this example, `proc_priocntl` is added to the default privilege set of user `db2user`.

Moreover, when the DB2 database manager is running in a non-global zone of Solaris, the `proc_priocntl` privilege must be added to the limit privilege set of the zone. To grant this privilege to the zone, logon as `root` and run the following command:

```
global# zonecfg -z db2zone
zonecfg:db2zone> set limitpriv="default,proc_priocntl"
```

In this example, `proc_priocntl` is added to the limit privilege set of the zone `db2zone`.

**Hard CPU shares**

The DB2 workload management dispatcher can manage CPU resources using shares-based entitlements that are assigned to service classes. Hard CPU shares, when assigned to a service class containing work viewed by the administrator as high impact or lower priority, prevents that service class from consuming more than its share of CPU resources whenever there is work in other service classes running on the system.

**Introduction**

Hard CPU shares can be assigned to any user and maintenance service class, but not to the system service class. After enabling the workload management dispatcher, monitoring your existing workloads to determine the extent of CPU resource consumption, and enabling the CPU shares attribute for service classes,
you can assign hard CPU shares to the service classes that you consider to be running lower-priority or high-impact work. Using hard CPU shares ensures that these service classes will have their CPU consumption limited in the presence of other workloads, both limiting their impacts on the system, and ensuring that the remaining CPU is reserved for other higher priority work.

The sections that follow describe the features and functionality of the hard CPU shares in more detail. A usage scenarios section helps to illustrate the hard CPU shares features and functionality with usage examples.

**Features and functionality**

When the host or logical partition (LPAR) is running at 100% CPU utilization, the allocation of CPU resources between service classes simply reflects their relative share percentages. On the other hand, when the host or LPAR begins to run below full CPU utilization, the reallocation of CPU resources is complex and dependent on whether the CPU shares attribute on each active service class is set to soft or hard CPU shares.

A service class with hard CPU shares assigned cannot exceed its CPU resource entitlement, indicated by its CPU shares configuration, to consume any unused CPU resources that become available on the host or LPAR. The workload management dispatcher always respects the CPU resource entitlement, determined by the relative amount of hard CPU shares that were assigned, when work is still running in competing service superclasses or running in competing service subclasses within the same service superclass. If competing workloads are not present or a competing workload temporarily becomes totally idle, service classes with hard CPU shares are able to claim unused CPU resources.

The hard CPU shares setting is most effective when used in cases where you want to strictly enforce the CPU resource entitlement on a service class to prevent work running in this service class from interrupting more important work running on the host or LPAR. Assign hard CPU shares to service classes that run complex or intensive queries that might otherwise degrade the performance of higher priority work due to contention on resources such as I/O, bufferpool, or CPU caches.

To enable the CPU shares attribute, you must set the value of the `wlm_disp_cpu_shares` database manager configuration parameter to `YES`. The default setting for this parameter is `NO`. After this parameter has been enabled, all existing and newly created service classes are assigned 1000 hard CPU shares by default to initially ensure an equal distribution of CPU resources. You can assign and adjust hard CPU shares by using the `CREATE SERVICE CLASS` and `ALTER SERVICE CLASS` statements. For complete details about how to enable and set hard CPU shares, see: “Enabling and setting CPU shares” on page 188.

Based on the number of CPU shares assigned to a service class, the workload management dispatcher calculates the proportion of the CPU resources that each service class is entitled to use. To determine the proportion of the CPU resources to which each service superclass is entitled, you can use the following formula to convert the number of CPU shares of a particular service superclass into a percentage of CPU resources allocated by the workload management dispatcher:

\[
\% \text{ CPU(superclass)} = \frac{\text{Number of superclass shares}}{\text{Total number of shares of all active superclasses}} \times 100
\]
To determine the proportion of the CPU resources to which each service subclass is entitled, you can use the following formula to convert the number of CPU shares of a particular service subclass into a percentage of CPU resources allocated by the workload management dispatcher:

\[
\% \text{CPU(subclass)} = \% \text{CPU(superclass)} \times \left( \frac{\text{Number of subclass shares}}{\text{Total number of shares of all active subclasses in the superclass}} \right)
\]

**Note:** The total number of CPU shares (both hard and soft) of all active superclasses are counted across all databases and all members on the host or LPAR.

**Usage scenarios**

**Scenario 1**

In Figure 30 on page 181 panel A, service class B has been assigned hard CPU shares and service classes A and C have been assigned soft CPU shares, the amounts of which are described in the figure legend. The pie chart represents the proportion of allocated CPU resources to which each of these active service classes are entitled and each service class is using their complete share of the CPU resources, therefore summing to 100% CPU utilization in this example. In panel B, service class A does not have enough work to fully use its CPU entitlement, dropping from 60% to 50% CPU utilization. The unused 10% of the CPU resources, temporarily relinquished by service class A, can be claimed by only the competing service class C based on its soft CPU shares assignment. Service class B cannot exceed its CPU resource allocation of 30% in this example because it has hard CPU shares assigned and there is enough work running in service classes A and C for those service classes to be considered active by the dispatcher. Panel C shows the total reallocation of CPU resources to service class C, increasing from 10% to 20% of the total available CPU resources.
If service class A experiences an increase in its workload, it effectively increases its demand on CPU resources. In this circumstance, service class C immediately relinquishes to service class A all of the claimed CPU resources, thereby restoring the state of CPU resource allocations to that which is depicted by the pie chart in panel A.

**Note:** When service classes compete to consume CPU resources, individual service class requests for CPU resources is handled by the workload management dispatcher in a first-come-first-served fashion. Due to typically frequent and short-lived requests for CPU resources on a busy host or LPAR, the reallocation of unused CPU resources over time results in a smooth redistribution of CPU resources which is in proportion to the relative CPU shares assignments.

**Scenario 2**

The use of hard shares can result in some portion of the CPU resources to remain under-utilized on the database server. Under-used CPU resources can occur in cases where the other service classes were not running a large enough workload to fully use their CPU resource allocations. Under-used CPU resources are desirable...
in cases where the hard shares are being used to limit an intensive workload that might otherwise interfere with the progress of other service classes, even when the CPU resources are below full utilization. This circumstance generally occurs due to contention on resources such as I/O or the CPU cache.

In Figure 31 on page 183 panel A, service classes B and C have been assigned hard CPU shares and service class A has been assigned soft CPU shares, the amounts of which are described in the figure legend. The pie chart represents the proportion of allocated CPU resources to which each of these active service classes are entitled and each service class is using their complete share of the CPU resources, therefore summing to 100% CPU utilization in this example. In panel B, service class A does not have enough work to fully use its CPU entitlement, dropping from 60% to 50% CPU utilization. The unused 10% of the CPU resources, temporarily relinquished by service class A, cannot be claimed by the competing service classes B and C based on their hard CPU shares assignment. Service classes B and C cannot exceed their CPU resource allocations of 30% and 10%, respectively, in this example because they both have hard CPU shares assigned and there is enough work running in service class A for it to be considered active by the dispatcher (CPU utilization falls below the level configured for the `wlm_disp_min_util` database manager configuration parameter; default is 5%). Panel C shows that the unused CPU resources are not reallocated and the CPU resources remain under-utilized in this scenario.
This scenario shows that you can protect the progress of work running in a high-priority service class from interruptions by work running in low-priority service classes.

**Scenario 3**

Hard CPU shares offer benefits over traditional fixed percentage CPU limits. In the absence of other workloads, the service class with hard CPU shares has the flexibility to claim unused CPU resources. Therefore, a service class with a hard CPU shares assignment is not artificially limited when other work is not present on the host or LPAR as that which occurs for a service class with fixed CPU limits.

In Figure 32 on page 184 panel A, service classes B and C have been assigned hard CPU shares and service class A has been assigned soft CPU shares, the amounts of which are described in the figure legend. The pie chart represents the proportion of allocated CPU resources to which each of these active service classes are entitled and each service class is using their complete share of the CPU resources, therefore summing to 100% CPU utilization in this example. In panel B, service class A does not have any work to fully use its CPU entitlement, dropping from 60% to 0%.
CPU utilization. The dispatcher considers service class A as being inactive. The unused 60% of the CPU resources, temporarily relinquished by service class A, can now be claimed by the competing service classes B and C based on their hard CPU shares assignment and an inactive service class. Service classes B and C, in this circumstance, can exceed their CPU resource allocations of 30% and 10%, respectively, because they both have hard CPU shares assigned and there is not enough work running in service class A for it to be considered active by the dispatcher (CPU utilization falls below the level configured for the `wlm_disp_min_util` database manager configuration parameter; default is 1%). Panel C shows that service class B is allocated 75% \((3000 / (3000 + 1000)) \times 100\) of the CPU resources and service class C is allocated 25% \((1000 / (3000 + 1000)) \times 100\).

If service class A experiences an increase in its workload, it effectively increases its demand on CPU resources. In this circumstance, service classes B and C immediately relinquish to service class A all of the claimed CPU resources, thereby restoring the state of CPU resource allocations to that which is depicted by the pie chart in panel A.
This scenario shows that although you can protect the progress of work running in high-priority service classes from interruptions by work running in low-priority service classes, when high-priority work is no longer present (such as during off-peak business hours), low-priority service classes with hard CPU shares still have the flexibility to claim the unused CPU resources.

**Note:** If either service class B or C does not use its full CPU resource allocation and is still considered by the dispatcher to be actively running work, the other service class with hard CPU shares cannot take advantage of it and use more than its allocation.

**More information**

Complete details are provided about the following workload management dispatcher subjects:

- Workload management dispatcher, see: “Workload management dispatcher” on page 159.
- Soft CPU shares, see: “Soft CPU shares.”
- CPU limit, see: “CPU limit” on page 190.
- Minimum CPU utilization level for a service class to be considered active, see: “Minimum CPU resource utilization for service class to be considered active” on page 205.
- Dispatch concurrency level, see: “Dispatch concurrency level” on page 208.
- Monitoring and tuning workload management dispatcher performance, see: “Monitoring and tuning workload management dispatcher performance” on page 210

**Soft CPU shares**

The DB2 workload management dispatcher can manage CPU resources using shares-based entitlements that are assigned to service classes. Soft CPU shares, when assigned to a service class by the administrator, give that service class the ability to consume more than its share of CPU resources if they are unused. When used in conjunction with other service classes bounded by hard CPU shares, this provides preferential treatment for that service class with regards to CPU resources.

**Introduction**

Soft CPU shares can be assigned to any user and maintenance service class, but not to the system service class. After enabling the workload management dispatcher and monitoring your existing workloads to determine the extent of CPU resource consumption, you can assign soft CPU shares to the service classes that are deemed high priority. Soft CPU shares are most effective when used on high priority workloads, because they allow a workload to consume more than its specified entitlement if there are any idle CPU resources on the system. Use of soft CPU shares is not recommended when you want to constrain the CPU consumption of lower priority or high-impact work; in this case, use hard CPU shares instead.

The sections that follow describe the features and functionality of the soft CPU shares in more detail. A usage scenario section helps to illustrate the soft CPU shares features and functionality with a usage example.
Features and functionality

When the host or logical partition (LPAR) is running at 100% CPU utilization, the allocation of CPU resources between service classes simply reflects their relative share percentages. On the other hand, when the host or LPAR begins to run below full CPU utilization, the reallocation of CPU resources is complex and dependent on whether the CPU shares attribute on each active service class is set to soft or hard CPU shares.

A service class with soft CPU shares assigned can exceed its CPU resource entitlement, indicated by its CPU shares configuration, to consume any unused CPU resources that become available on the host or LPAR. When two or more service classes have soft shares and unused CPU resources become available with enough CPU resource demand from each service class to consume the spare capacity, allocation of the CPU resources to the competing service classes is done proportionally according to the relative share of each active service class. The soft CPU shares setting is most effective for high-priority work that you want to be able to temporarily claim any spare CPU resources that become available. In addition, the soft CPU shares setting is most effective for workloads consisting of short queries that are expected to have a relatively modest impact on database resources, outside of their immediate CPU consumption.

To enable the CPU shares attribute, you must set the value of the `wlm Disp_cpu_shares` database manager configuration parameter to YES. The default setting for this parameter is NO. You can assign and adjust soft CPU shares by using the CREATE SERVICE CLASS and ALTER SERVICE CLASS statements. For complete details about how to enable and set soft CPU shares, see: “Enabling and setting CPU shares” on page 188.

Based on the number of CPU shares assigned to a service class, the workload management dispatcher calculates the proportion of the CPU resources that each service class is entitled to use. To determine the proportion of the CPU resources to which each service superclass is entitled, you can use the following formula to convert the number of CPU shares of a particular service superclass into a percentage of CPU resources allocated by the workload management dispatcher:

\[
\% \text{CPU(superclass)} = \frac{\text{Number of superclass shares}}{\text{Total number of shares of all active superclasses}} \times 100
\]

To determine the proportion of the CPU resources to which each service subclass is entitled, you can use the following formula to convert the number of CPU shares of a particular service subclass into a percentage of CPU resources allocated by the workload management dispatcher:

\[
\% \text{CPU(subclass)} = \% \text{CPU(superclass)} \times \frac{\text{Number of subclass shares}}{\text{Total number of shares of all active subclasses in the superclass}}
\]

Note: The total number of CPU shares (both hard and soft) of all active superclasses are counted across all databases and all members on the host or LPAR.

Usage scenario

In Figure 33 on page 187 panel A, service classes A, B, and C have been assigned soft CPU shares, the amounts of which are described in the figure legend. The pie chart represents the proportion of allocated CPU resources to which each of these active service classes are entitled and each service class is using their complete share of the CPU resources, therefore summing to 100% CPU utilization in this
example. In panel B, service class A does not have enough work to fully use its CPU entitlement, dropping from 60% to 50% CPU utilization. The unused 10% of the CPU resources, temporarily relinquished by service class A, can be proportionally claimed by the competing service classes B and C based on their relative soft CPU shares assignments. Panel C depicts the proportional reallocation of CPU resources between service classes B and C, with service class B getting 7.5% \( (10\% \times (3000/4000)) \) and service class C getting 2.5% \( (10\% \times (1000/4000)) \) of the unused 10% relinquished by service class A.

![Legend](chart.png)

Figure 33. Soft CPU shares pie charts

If service class A experiences an increase in its workload, it effectively increases its demand on CPU resources. In this circumstance, service classes B and C immediately relinquish to service class A all of the claimed CPU resources, thereby restoring the state of CPU resource allocations to that which is depicted by the pie chart in panel A.

**Note:** When service classes compete to consume CPU resources, individual service class requests for CPU resources is handled by the workload management dispatcher in a first-come-first-served fashion. Due to typically frequent and
short-lived requests for CPU resources on a busy host or LPAR, the reallocation of unused CPU resources over time results in a smooth redistribution of CPU resources which is in proportion to the relative CPU shares assignments.

**More information**

Complete details are provided about the following workload management dispatcher subjects:
- Workload management dispatcher, see: “Workload management dispatcher” on page 159.
- Hard CPU shares, see: “Hard CPU shares” on page 178.
- CPU limit, see: “CPU limit” on page 190.
- Minimum CPU utilization level for a service class to be considered active, see: “Minimum CPU resource utilization for service class to be considered active” on page 205.
- Dispatch concurrency level, see: “Dispatch concurrency level” on page 208.
- Monitoring and tuning workload management dispatcher performance, see: “Monitoring and tuning workload management dispatcher performance” on page 210

**Enabling and setting CPU shares**

Enable the CPU shares attributes by setting the value of the `wlm_disp_cpu_shares` database manager configuration parameter to `YES`. Set the hard or soft CPU shares attribute using the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement. You will generally want to set hard CPU shares on service classes running low-priority or high-impact work, and soft CPU shares on high-priority service classes when CPU resources are strained.

**Before you begin**

**Note**: For the CPU shares setting to be effective, you must enable the workload management dispatcher by configuring the `wlm_dispatcher` database manager configuration parameter. This can be done either before or after you have enabled and set the CPU shares. See: “Enabling workload management dispatcher” on page 177.

**About this task**

The CPU shares attributes are enabled in this task by using the command line processor method, and the CPU shares attribute is set for a service class by using the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement.

**Restrictions**

You can assign soft CPU shares only to service classes that the workload management dispatcher can control, which are user and maintenance service classes. The CPU resource allocations for system service classes cannot be controlled by the workload management dispatcher.

**Procedure**

Skip to step 2 if the CPU shares attributes have already been enabled.
To enable the CPU shares attributes:
1. Using the command line processor (CLP), issue the **UPDATE DATABASE MANAGER CONFIGURATION** (or **UPDATE DBM CFG**) command and set the value of the **wlm_disp_cpu_shares** database manager configuration parameter to **YES**; the parameter value is updated immediately by first attaching to your DB2 instance:

```
attach to instance-name
update dbm cfg using wlm_disp_cpu_shares yes
detach
```

Skip to step 3 if an existing service class hard or soft CPU shares setting is to be altered.

To create a new service class and set the hard or soft CPU shares attribute:

2. Issue the **CREATE SERVICE CLASS** statement to create a new service class and set the hard or soft CPU shares value to 5000 in this example:

```
create service class service-class-name hard cpu shares 5000
create service class service-class-name soft cpu shares 5000
```

To alter the hard or soft CPU shares setting for an existing service class:

3. Issue the **ALTER SERVICE CLASS** statement to alter the hard or soft CPU shares value to 15000 in this example:

```
alter service class service-class-name hard cpu shares 15000
alter service class service-class-name soft cpu shares 15000
```

**Results**

The CPU shares attributes have been enabled and you have set or altered the hard or soft CPU shares setting for the specified service class. Service classes assigned hard CPU shares can, under restricted conditions, claim CPU resources that other service classes have relinquished due to their lessened workload demand. Service classes assigned soft CPU shares, on the other hand, can always claim CPU resources that other service classes have relinquished due to their lessened workload demand.

**What to do next**

After having enabled the workload management dispatcher to manage CPU resources, having enabled the CPU shares attributes, and set the hard or soft CPU shares attribute, consider completing the following tasks:

- For the hard or soft CPU shares setting to be effective, you must enable the workload management dispatcher by configuring the **wlm_dispatcher** database manager configuration parameter. See: “Enabling workload management dispatcher” on page 177.
- Set the dispatch concurrency level by configuring the **wlmDispConcur** database manager configuration parameter. See: “Setting dispatch concurrency level” on page 209.
- Create or alter a service class and configure the CPU limit by using the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement. See: “Setting a CPU limit” on page 204.
- Set the minimum CPU utilization percentage for the workload management dispatcher to consider service classes to be active by configuring the **wlmDispMinUtil** database manager configuration parameter. See: “Setting minimum CPU resource utilization for service class to be considered active” on page 207.
CPU limit

The DB2 workload management dispatcher can enforce fixed CPU limit entitlements that can be assigned to service superclasses and subclasses. By applying a CPU limit, you can limit the CPU that is consumed by a service class to a fixed amount on the system, regardless of any other work running in the DB2 database manager. This leaves the remaining portion of CPU resources available for other consumers to use. When CPU limits are used in conjunction with CPU shares, the most limiting or restrictive condition is always honored.

Introduction

A CPU limit can be assigned to any user and maintenance service class, but not to the system service class. After having enabled the workload management dispatcher and having monitored your existing workloads to determine the extent of CPU resource consumption, you can then assign CPU limits to the service classes that you want to be strictly limited in their CPU consumption.

CPU shares provide you with the ability to control the CPU resource entitlement of individual workloads when the overall workload of the host or LPAR is heavy and yet not waste CPU resources when the overall workload is light. However, there are workloads for which you always want to limit their CPU resource entitlement despite the light overall workload of the host or LPAR. For example, if multiple departments share the cost of purchasing a database server, each department might want to ensure that the other departments are not taking more than their allocated CPU resource entitlements, despite the possibility that the selected configuration can result in under-utilizing the CPU resources of the host or LPAR. CPU shares do not give you this level of control, whereas a CPU limit does.

The sections that follow describe the features and functionality of the CPU limit in more detail. A usage scenario section helps to illustrate the CPU limit features and functionality with usage examples.

Features and functionality

The CPU limit gives you the ability to place a fixed limit on the CPU resource entitlement by work in a service class. If a CPU limit is set on all service classes, you can reserve a portion of the CPU resources to perform work regardless of any other work running in the DB2 database manager. Configuring a CPU limit on a service class effectively provides a strictly-enforced sandbox for your workloads in which you can achieve fairness in CPU resource consumption between workloads, but at the expense of occasionally not achieving full utilization of the CPU resources.

The CPU limit is also useful when you have multiple DB2 instances on your host or LPAR. With the workload management dispatcher operating at the instance level, the CPU resource allocation of any service class is computed from the shares of that service class relative to the shares of all other service classes within the instance. A CPU limit, however, is expressed as a percentage of the CPU resources of the host or LPAR, regardless of how many DB2 instances exist on such a host or LPAR. By applying CPU limits to your superclasses and shares to your subclasses, you can use the CPU limit to control the absolute CPU resource entitlement of each superclass, and by extension the instance, and then use shares to control the relative CPU resource entitlements of service subclasses running within those superclasses.
A CPU limit can be configured both at the service superclass level, where it represents a percentage limit of CPU resource entitlement on the host or LPAR by all subclasses in that superclass, or at the subclass level, where it represents a percentage limit of CPU resource entitlement on the host or LPAR by that particular subclass.

To enable the CPU limit attribute, you must enable the workload management dispatcher by setting the value of the `wlm.dispatcher` database manager configuration parameter to `ON`. The default setting for this parameter is `OFF`. By enabling the workload management dispatcher, CPU resource control using the CPU limit attribute becomes available by default. You can assign and adjust CPU limits by using the `CREATE SERVICE CLASS` and `ALTER SERVICE CLASS` statements. For complete details about how to set CPU limits, see: “Setting a CPU limit” on page 204.

The workload management dispatcher always respects the most restrictive CPU limit or CPU shares assignments when allocating CPU resources to service classes. For example, when a CPU limit is set at both the superclass and subclass levels, the more restrictive CPU limit is honored. Likewise, if a service class reaches its CPU limit before it has fully utilized its shares-based CPU resource entitlement, the dispatcher respects the CPU limit.

**Usage scenarios**

**CPU limit and multiple superclasses**

This set of usage examples addresses CPU limit behavior in a multiple superclass environment.

Figure 34 on page 192 shows a host or LPAR configured with two superclasses, A and B. For illustration purposes to help describe the basic concepts, assume that there is negligible work running in the default user, maintenance, and system service classes. For the following scenarios, there is only one DB2 instance with one database and only one member on this host or LPAR.
CPU limit and multiple superclasses: Scenario 1

In this example, Figure 35 on page 193 panel A shows that service class A has a CPU limit of 30% and service class B has no CPU limit. At the start of this scenario, service class A has at least enough work to drive its CPU resource utilization to 30% and service class B has at least enough work to drive its CPU resource utilization to 70%. Service classes A and B both have 1000 soft CPU shares.
Panel B shows that service class A has a reduction in CPU resource demand from 30% to 20%. Service class B has more than enough CPU resource demand to claim the CPU resources temporarily relinquished by service class A, as shown in panel C.

**CPU limit and multiple superclasses: Scenario 2**

In this example, Figure 36 on page 194 panel A again shows that service class A has a CPU limit of 30% and service class B has no CPU limit. At the start of this scenario, service class A has at least enough work to drive its CPU resource utilization to 30% and service class B has at least enough work to drive its CPU resource utilization to 70%. Service classes A and B both have 1000 soft CPU shares.
Panel B shows that service class B has a reduction in CPU resource demand from 70% to 50%. Due to its CPU limit, service class A cannot consume the 20% of the CPU resources that service class B temporarily relinquished. The total CPU utilization for the host or LPAR remains at 80%.

CPU limit and multiple superclasses: Scenario 3

In this example, Figure 37 on page 195 panel A again shows that service class A has a CPU limit of 30% and service class B has no CPU limit. At the start of this scenario, service class A has at least enough work to drive its CPU resource utilization to 30% and service class B has at least enough work to drive its CPU resource utilization to 70%. Service classes A and B both have 1000 soft CPU shares.
Panel B shows that service class B has a reduction in CPU resource demand from 70% to 0%. Due to its CPU limit, service class A cannot consume the 70% of the CPU resources that service class B temporarily relinquished. The total CPU utilization for the host or LPAR remains at 30%.

CPU limit and multiple subclasses

This next set of usage examples addresses CPU limit behavior in a multiple subclass environment.

Figure 38 on page 196 shows a host or LPAR configured with two superclasses, A and B. Inside service superclass A are service subclasses A1 and A2. For illustration purposes to help describe the basic concepts, assume that there is negligible work running in the default user, maintenance, and system service classes. For the following scenarios, there is only one DB2 instance with one database and only one logical partition on this host or LPAR.
CPU limit and multiple subclasses: Scenario 1

In this example, Figure 39 on page 197 panel A shows that service superclass class A has a CPU limit of 50% and service subclass A1 has a CPU limit of 20%. Service superclass B has no CPU limit. At the start of this scenario, service subclass A1 has at least enough work to drive its CPU resource utilization to 20% and service subclass A2 has at least enough work to drive its CPU resource utilization to 30%, giving service superclass A a total CPU resource utilization of 50%. Service superclass B has at least enough work to drive its CPU resource utilization to 50%. Service superclasses A and B both have 1000 soft CPU shares.

Figure 38. Data server configuration: Multiple subclasses
Panel B shows that service subclass A1 has a reduction in CPU resource demand from 20% to 10%. Service subclass A2 and service superclass B each have more than enough CPU demand to claim the entire amount of unused CPU resources that service class A1 has temporarily relinquished. However, service subclass A2 is allocated all of the relinquished CPU resource and increases its CPU utilization from 30% to 40%, as depicted in panel C. This CPU resource allocation result is due to the service superclasses A and B already sharing the total CPU resources with a 50%/50% equal split resulting from the 1000 soft CPU shares assigned to each of the superclasses.

CPU limit and multiple subclasses: Scenario 2

In this example, Figure 40 on page 198 panel A again shows that service superclass class A has a CPU limit of 50% and service subclass A1 has a CPU limit of 20%. Service superclass B has no CPU limit. At the start of this scenario, service subclass A1 has at least enough work to drive its CPU resource utilization to 20% and service subclass A2 has at least enough work to drive its CPU resource utilization to 30%, giving service superclass A a total CPU resource utilization of 50%. Service superclass B has at least enough work to drive its CPU resource utilization to 50%. Service superclasses A and B both have 1000 soft CPU shares.
Panel B shows that service subclass A2 has a reduction in CPU resource demand from 30% to 20%. Due to service subclass A1 having a CPU limit of 20%, service subclass A1 cannot exceed its current 20% CPU utilization. Service superclass A drops its CPU utilization to a total of 40%. Service superclass B has enough CPU resource demand to claim the CPU resource that was temporarily relinquished by service subclass A2, increasing the CPU utilization of service superclass B from 50% to 60%, as shown in panel C.

**CPU limit and multiple DB2 instances**

This next set of usage examples addresses CPU limit behavior in a multiple DB2 instance environment.

Figure 41 on page 199 shows a host or LPAR configured with two DB2 instances, instance A and instance B. Each instance contains one database, database 1 and database 2. Each database contains one service superclass, service superclass A and B. Inside service superclass A are service subclasses A1 and A2. Inside service superclass B are service subclasses B1 and B2. To keep the scenario simple, user requests for each database are routed to only the two service subclasses and no work is routed to the default service subclass. In addition, to help describe the basic concepts, assume that there is negligible work running in the default user,
maintenance, and system service classes. For the following scenarios, there are two DB2 instances with one database in each instance and only one member on this host or LPAR.

**CPU limit and multiple DB2 instances: Scenario 1**

In this example, instance A is to receive no more than 50% of the total CPU resources of the host or LPAR and instance B is to receive no more than 50% of the total CPU resources of the host or LPAR. Within instance A, 80% of the instance CPU resources are to be allocated to service subclass A1 and 20% to service subclass A2 by assigning 8000 soft CPU shares to A1 and 2000 soft CPU shares to A2. When service subclass A1 does not consume its full 80%, we want service subclass A2 to claim any unused CPU resources, and vice versa, to maximize CPU utilization within instance A. Within instance B, 60% of the instance CPU resources are to be allocated to service subclass B1 and 40% to service subclass B2 by assigning 6000 soft CPU shares to B1 and 4000 soft CPU shares to B2. As with service subclasses A1 and A2, we want to maximize CPU utilization within instance B by having service subclass B1 claim any unused CPU resources when service subclass B2 does not use its full CPU resource entitlement, and vice versa.

To configure the appropriate conditions for this scenario as described in the preceding paragraph, limit the CPU resources for each instance by creating a CPU limit of 50% on service superclass A, in database 1, in instance A, and creating a CPU limit of 50% on service superclass B, in database 2, in instance B. When both instances are using all of their CPU resource entitlements, the host or LPAR as a whole is considered to have 100% CPU utilization. If either instance does not use its full CPU entitlement, the other instance cannot claim the unused CPU resources.
For each service superclass on each instance, the workload management dispatcher divides the available CPU resources among the subclasses using their relative soft CPU shares assignments. By using soft CPU shares assigned to the service subclasses, the only circumstance in which an instance does not make full use of their CPU resource entitlement occurs when every subclass in the instance does not have enough work running within it to achieve full CPU utilization of its entitlement.

Figure 42 on page 201 panel A shows that service superclass class A has a CPU limit of 50% and service superclass B has a CPU limit of 50%. Service subclass A1 has 80% of the instance A CPU resources (40% of the host or LPAR CPU resources); service subclass A2 has the remaining 20% (10% of the host or LPAR CPU resources). Service subclass B1 has 60% of the instance B CPU resources (30% of the host or LPAR CPU resources); service subclass B2 has the remaining 40% (20% of the host or LPAR CPU resources). At the start of this scenario, service subclass A1 has at least enough work to drive its host or LPAR CPU resource utilization to 40% and service subclass A2 has at least enough work to drive its host or LPAR CPU resource utilization to 10%, giving service superclass A a total CPU resource utilization of 50%. Service subclass B1 has at least enough work to drive its host or LPAR CPU resource utilization to 30% and service subclass B2 has at least enough work to drive its host or LPAR CPU resource utilization to 20%, giving service superclass B a total CPU resource utilization of 50%.
Panel B shows that service subclass A1 has a reduction in CPU resource demand from 40% to 30% and service subclass B1 has a reduction in CPU resource demand from 30% to 20%. Assuming service subclass A2 has enough work executing within it to claim any unused CPU resources that become available, A2 increases its CPU utilization from 10% to 20% because A2 has soft CPU shares assigned to it, as shown in panel C. The same applies to service subclass B2 which increases its CPU utilization from 20% to 30%, as shown in panel C.

Let's now consider the original start condition as described in panel A for a different example. If service subclass B1 reduces its CPU demand from 30% to 20% and service subclass B2 does not have enough work running within it to exceed a CPU demand of 20%, then service superclass B in instance B does not use its full CPU entitlement of 50% and remains at a CPU utilization of 40%, as shown in panel D. The result is that the host or LPAR has a CPU utilization of only 90% of the total CPU resources.

CPU limit and multiple DB2 instances: Scenario 2

This scenario example illustrates what happens when the sum of the CPU limits of the service subclasses does not exceed the CPU limit of the parent service superclass.
Using similar initial conditions as in “CPU limit and multiple DB2 instances: Scenario 1”, let's just change the soft CPU shares assigned to service subclasses A1 and A2 into CPU limits of 40% and 10%, respectively, of the total CPU resources for the host or LPAR, as shown in Figure 43 panel A. When both service subclasses use the CPU resources up to their assigned CPU limits, the total CPU utilization is 50% for service superclass A, making redundant the additional constraint of a 50% CPU limit on service superclass A.

Panel B shows that service subclass A1 has a reduction in CPU resource demand from 40% to 30% due to a decrease in the amount of work running in the service subclass. In this circumstance, the dispatcher cannot allocate the unused CPU resource, temporarily relinquished by service subclass A1, to service subclass A2. Service subclass A2 continues running workloads on its CPU limit of 10% of the CPU resources of the host or LPAR. This situation makes instance A unable to use its full CPU resource entitlement of 50%.

**CPU limit and multiple DB2 instances: Scenario 3**

This scenario example illustrates what happens when the sum of the CPU limits of the service subclasses exceed the CPU limit of the parent service superclass.

Using similar initial conditions as in “CPU limit and multiple DB2 instances: Scenario 1”, let’s just change the soft CPU shares assigned to service subclasses A1 and A2 into CPU limits of 40% and 40%, respectively, of the total CPU resources for the host or LPAR. In this example, the total of the assigned CPU limits for service subclasses A1 and A2 is 80% which exceeds the 50% CPU limit assigned to service superclass A. The workload management dispatcher prevents service superclass A from exceeding its 50% CPU limit. The amount of CPU resources that is allocated to each service subclass in superclass A is determined by the CPU shares that have been assigned to the subclasses. CPU shares were not explicitly assigned for service subclasses A1 and A2, but each of these subclasses has the 1000 soft CPU shares that were assigned when the subclasses were created, giving
each subclass an equal CPU entitlement. The dispatcher allocates an equal division of the 50% of the total host or LPAR CPU resources entitled to service superclass A. The result is that 25% is allocated to service subclass A1 and 25% is allocated to service subclass A2, as shown in Figure 44 panel A.

![Diagram](image)

Figure 44. CPU limit and multiple DB2 instances: Scenario 3

Panel B shows that service subclass A1 has a reduction in CPU resource demand from 25% to 5% due to a decrease in the amount of work running in the service subclass. Due to the soft CPU shares assignment, the dispatcher can allocate the unused CPU resource, temporarily relinquished by service subclass A1, to the point of 40% claimed by service subclass A2. Service subclass A2 is unable to exceed its CPU limit of 40% of the CPU resources of the host or LPAR. This situation makes instance A unable to use its full CPU resource entitlement of 50%.

**Note:** The limitations of this approach to managing work in multiple instances on the same host or LPAR is that it limits you to one service superclass in each instance. For operating systems that have a workload manager into which DB2 workload management can be integrated, an alternative is to map the DB2 service classes of each instance to a service class in an operating system (OS) WLM (such as AIX WLM and Linux WLM) and assign the hard limits of the OS WLM on each OS service class to put upper bounds on the CPU resource utilization of each instance.

**More information**

Complete details are provided about the following workload management dispatcher subjects:

- Workload management dispatcher, see: “Workload management dispatcher” on page 159.
- Dispatch concurrency level, see: “Dispatch concurrency level” on page 208.
Setting a CPU limit

Set the CPU limit attribute using the CREATE SERVICE CLASS statement to create a new service class and CPU limit, and use the ALTER SERVICE CLASS statement to alter the CPU limit setting for an existing service class. You set the CPU limit to cap CPU utilization by a service class under all circumstances during workload management dispatcher control.

Before you begin

You must enable the workload management dispatcher by configuring the wlm_dispatcher database manager configuration parameter, if not already done. By enabling the workload management dispatcher, CPU resource control using the CPU limit attribute becomes available by default. See: “Enabling workload management dispatcher” on page 177.

About this task

The CPU limit is set for a service class in this task using the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement using the command line processor.

Restrictions

You can assign CPU limits only to service classes that the workload management dispatcher can control, which are user and maintenance service classes. The CPU resource allocations for system service classes cannot be controlled by the workload management dispatcher.

Procedure

To create a new service class and set the CPU limit (skip to step 2 if an existing service class CPU limit setting is to be altered):

1. Issue the CREATE SERVICE CLASS statement to create a new service class and set the CPU limit to 25% in this example:

   ```
   create service class service-class-name cpu limit 25
   ```

To alter the CPU limit setting for an existing service class:

2. Issue the ALTER SERVICE CLASS statement to alter the CPU limit to 50% in this example:

   ```
   alter service class service-class-name cpu limit 50
   ```

Results

You have set or altered the CPU limit setting for the specified service class. The workload management dispatcher cannot allocate CPU resources to this service class beyond the set limit.
What to do next

After having enabled the workload management dispatcher to manage CPU resources and having set the CPU limit, consider completing the following tasks:

- Set the dispatch concurrency level by configuring the `wlm_disp_concur` database manager configuration parameter. See: “Setting dispatch concurrency level” on page 209.
- Enable hard and soft CPU shares by configuring the `wlm Disp_cpu_shares` database manager configuration parameter. See step 1 in: “Enabling and setting CPU shares” on page 188.
- Create or alter a service class and configure a hard or soft CPU share by using the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement. See: “Enabling and setting CPU shares” on page 188.
- Set the minimum CPU utilization percentage for the workload management dispatcher to consider service classes to be active by configuring the `wlm_disp_min_util` database manager configuration parameter. See: “Setting minimum CPU resource utilization for service class to be considered active” on page 207.

Minimum CPU resource utilization for service class to be considered active

By setting the `wlm_disp_min_util` database manager configuration parameter, you are able to control the minimum level of CPU resource utilization at which the workload management dispatcher considers the service class actively engaged in executing work. The CPU shares of only the active service classes are factored into CPU resource allocation scheduling that is performed by the dispatcher.

When managing shares-based CPU resource allocations, the workload management dispatcher considers a service class to be active and factors its entire CPU shares assignment in the overall CPU resource scheduling allocation if any amount of the CPU resource is being used by database requests executing in that service class. In certain instances, it is desirable to have some control over how much service-class-generated CPU activity is needed for the workload management dispatcher to include the CPU shares of the service class during CPU resource scheduling.

In the example scenario depicted in Figure 45 on page 206 panel A, service class A contains a high-priority transactional workload that runs only during the business day, while service class B and C contain ongoing day and night, low-priority batch jobs. Service class A is protected from interruption by assigning hard CPU shares to service classes B and C. At night when the transactional workload on service class A is not present (as shown in panel B), service classes B and C are able to make full use of the CPU resources and make much faster progress (as shown in panel C).
Let's now consider what happens if a small trickle of transactional work continues to occur at night on service class A. In this case, service class A is considered to be active by the workload management dispatcher, and the unused CPU resources temporarily relinquished by service class A are not available to service classes B and C. The CPU resource entitlements for service classes B and C are as shown in panel B with a small sliver of the CPU utilization pie representing the activity of service class A (not shown in the panel B pie chart). Much slower overnight progress is the result for service classes B and C than that made in the original scenario as shown in panel C.

Maximum user flexibility to manage their DB2 workloads is provided by the option to set a percentage of CPU utilization at or above which a service class is considered to be active on the host or LPAR. When a service class is considered to be inactive, its CPU shares assignment is not factored into CPU resource entitlement calculations, thereby allowing service classes, particularly those with assigned hard CPU shares, to claim the unused CPU resources. This minimum percentage of CPU utilization is specified by configuring the percentage value of the `wlm_disp_min_util` database manager configuration parameter. This configuration parameter can be set to a percentage value between 0 and 100, with
a default value of 5. For this configuration parameter setting to be effective, the workload management dispatcher must be enabled by setting the value of the \texttt{wlm\_dispatcher} database manager configuration parameter to \texttt{YES}.

Let's reconsider the small trickle scenario described earlier. With the percentage value of the \texttt{wlm\_disp\_min\_util} database manager configuration parameter now set to a value slightly higher than the small sliver of overnight CPU utilization for service class A, we can be more confident that the progress of the overnight batch jobs of service classes B and C is much improved, similar to that of the original scenario in which the CPU utilization was as that shown in panel C.

More information

Complete details are provided about the following workload management dispatcher subjects:

- Workload management dispatcher, see: "Workload management dispatcher" on page 159.
- Dispatch concurrency level, see: "Dispatch concurrency level" on page 208.
- Hard CPU shares, see: "Hard CPU shares" on page 178.
- Soft CPU shares, see: "Soft CPU shares" on page 185.
- CPU limit, see: "CPU limit" on page 190.
- Monitoring and tuning workload management dispatcher performance, see: "Monitoring and tuning workload management dispatcher performance" on page 210

Setting minimum CPU resource utilization for service class to be considered active

Set the minimum CPU utilization for a service class by setting the value of the \texttt{wlm\_disp\_min\_util} database manager configuration parameter to a percentage. Service classes utilizing CPU resources equal to or greater than that percentage are considered active on the host or logical partition (LPAR) and the CPU shares of the active service classes are factored into CPU allocation calculations.

Procedure

To set the minimum CPU utilization percentage for service classes to be considered active:

Using the command line processor (CLP), issue the \texttt{UPDATE DATABASE MANAGER CONFIGURATION} (or \texttt{UPDATE DBM CFG}) command and set the value of the \texttt{wlm\_disp\_min\_util} database manager configuration parameter to 6%; the parameter value is updated immediately by first attaching to your DB2 instance:

\begin{verbatim}
attach to instance-name
update dbm cfg using wlm_disp_min_util 6
detach
\end{verbatim}

Results

The minimum CPU utilization percentage for service classes to be considered active has been set. Service classes that have a CPU utilization percentage that is less than this minimum are considered to be idle and their CPU allocations can then be proportionally reallocated to service classes that can claim spare CPU resources.
What to do next

After having set the minimum CPU utilization percentage for service classes to be considered active by the workload management dispatcher, consider completing the following tasks:

- For the minimum CPU utilization setting to be effective, you must enable the workload management dispatcher by configuring the `wlm_dispatcher` database manager configuration parameter. See: “Enabling workload management dispatcher” on page 177.
- Set the dispatch concurrency level by configuring the `wlm_disp_concur` database manager configuration parameter. See: “Setting dispatch concurrency level” on page 209.
- Create or alter a service class and configure the CPU limit by using the `CREATE SERVICE CLASS` or `ALTER SERVICE CLASS` statement. See: “Setting a CPU limit” on page 204.
- Enable hard and soft CPU shares by configuring the `wlm_disp_cpu_shares` database manager configuration parameter. See step 1 in: “Enabling and setting CPU shares” on page 188.
- Create or alter a service class and configure a hard or soft CPU share by using the `CREATE SERVICE CLASS` or `ALTER SERVICE CLASS` statement. See: “Enabling and setting CPU shares” on page 188.

Dispatch concurrency level

The workload management dispatcher controls the number of threads that the database manager will allow to be simultaneously dispatched to the operating system (OS). This number is referred to as the dispatch concurrency level. You can set the dispatch concurrency level using the `wlm_disp_concur` database manager configuration parameter. Details are provided here.

Features and functionality

The number of simultaneously executing agents managed by the workload management dispatcher should generally be a low multiple of the number of CPUs on the system. The goal is to establish a concurrency level just large enough to ensure that the CPU resources on the host or LPAR are fully utilized and no larger, while accounting for OS scheduling latencies as threads switch from being active to inactive and vice versa. This optimal value ensures maximum efficiency and gives the workload management dispatcher maximum control over CPU resource allocation. If the configured concurrency value is too small, there will not be enough work to fully utilize the available CPU resources. If the configured concurrency value is too large, the dispatcher will have less control over the work running on the system, which might add to resource contention.

By setting the `wlm_disp_concur` database manager configuration parameter, you can either specify how the DB2 database manager itself sets the concurrency level (`COMPUTED`), or you can manually set the concurrency level to a fixed value.

As a general guideline, the optimal dispatch concurrency level is four times the number of CPUs available on the server hardware. This guideline is typically sufficient for most platforms and is the level you get as the default (`COMPUTED` is the default setting for the configuration parameter). For complete details about how to set the dispatch concurrency level, see: “Setting dispatch concurrency level” on page 209.
The specified value of the \texttt{wlm\_disp\_concur} database configuration parameter applies to each host or LPAR. In multimember database environments, the specified concurrency level is shared between the members.

**Tuning**

After the workload management dispatcher has been enabled, monitor your system for a drop in throughput and CPU utilization. If a drop does occur in either metric, increase the value of the configuration parameter in increments of the number of available processors.

**More information**

Complete details are provided about the following subjects:

- Workload management dispatcher, see: “Workload management dispatcher” on page 159.
- Soft CPU shares, see: “Soft CPU shares” on page 185.
- Hard CPU shares, see: “Hard CPU shares” on page 178.
- CPU limit, see: “CPU limit” on page 190.
- Minimum CPU utilization level for a service class to be considered active, see: “Minimum CPU resource utilization for service class to be considered active” on page 205.
- Monitoring and tuning workload management dispatcher performance, see: “Monitoring and tuning workload management dispatcher performance” on page 210

**Setting dispatch concurrency level**

Set the dispatch concurrency level by setting the value of the \texttt{wlm\_disp\_concur} database manager configuration parameter to \texttt{COMPUTED} or a manually-set value. You set the dispatch concurrency level to an optimal value that ensures maximum efficiency and that gives the workload management dispatcher maximum control over CPU resource allocations.

**Procedure**

To set the dispatch concurrency level:

Using the command line processor (CLP), issue the \texttt{UPDATE DATABASE MANAGER CONFIGURATION} (or \texttt{UPDATE DBM CFG}) command and set the value of the \texttt{wlm\_disp\_concur} database manager configuration parameter to one of the following possibilities:

- \texttt{COMPUTED} (default)
- \texttt{manually\_set\_value}

The following example manually sets the dispatch concurrency level to 16, which is 4 times the 4 CPUs available on the system, and is updated immediately by first attaching to your DB2 instance:

```
attach to instance-name
update dbm cfg using wlm\_disp\_concur 16
detach
```

**Results**

The dispatch concurrency level has been set. If set to \texttt{COMPUTED}, the concurrency level is determined by the DB2 database manager. If manually set, you might need
to tune the concurrency level to an optimal value that ensures maximum efficiency and that gives the workload management dispatcher maximum control over CPU resource allocations.

**What to do next**

After having set the dispatch concurrency level, consider completing the following tasks:

- For the dispatch concurrency level setting to be effective, you must enable the workload management dispatcher by configuring the `wlm_dispatcher` database manager configuration parameter. See: “Enabling workload management dispatcher” on page 177.
- Create or alter a service class and configure the CPU limit by using the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement. See: “Setting a CPU limit” on page 204.
- Enable hard and soft CPU shares by configuring the `wlm_disp_cpu_shares` database manager configuration parameter. See step 1 in: “Enabling and setting CPU shares” on page 188.
- Create or alter a service class and configure a hard or soft CPU share by using the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement. See: “Enabling and setting CPU shares” on page 188.
- Set the minimum CPU utilization percentage for the workload management dispatcher to consider service classes to be active by configuring the `wlm_disp_min_util` database manager configuration parameter. See: “Setting minimum CPU resource utilization for service class to be considered active” on page 207.

After the workload management dispatcher has been enabled and the dispatch concurrency level has been set, monitor your system for a drop in throughput and CPU utilization. If a drop in either metric does occur, increase the value of the `wlm_disp_concur` configuration parameter in increments of the number of available processors. For more information about monitoring and tuning workload management dispatcher performance, see: “Monitoring and tuning workload management dispatcher performance”

**Monitoring and tuning workload management dispatcher performance**

The monitoring and tuning of workload management dispatcher performance can be achieved with the table functions and monitoring elements provided with the DB2 database manager. Details are provided here.

**Introduction**

To monitor and tune the performance of the workload management dispatcher and achieve the best results, you require the proper tools. Table functions and monitoring elements are provided that can help you monitor the performance of the dispatcher. After analyzing the collected monitoring data as described here, you can adjust the dispatcher concurrency level or redistribute CPU entitlements by adjusting service class CPU shares and CPU limits to tune the dispatcher performance.

The following sections describe the types of workloads to consider, because they differ in how they are best monitored to deliver the appropriate data for you to analyze, and the performance measures which are the most suitable for the
particular type of workload under consideration.

**Types of workloads**

From the perspective of measuring performance for the purpose of tuning your dispatcher configuration to obtain the best possible workload performance from your system, there are two types of workloads to consider: batch and transactional. Each type of workload has characteristic measures of performance that are best suited to determining how well your system is performing under that type of workload. Use the performance measures that best characterize the workload type your system is experiencing.

**Batch**

A batch workload has one or more applications connecting to the database and each application submits activity after activity or transaction after transaction without any pause. The most important measure of the performance of this workload is how quickly the entire set of activities or transactions is completed. The processing speed of the database manager is the main determinant of how quickly the entire set of activities or transactions is completed.

**Transactional**

A transactional workload has a user at a terminal that submits an activity or transaction to the database, then waits for a response, analyzes the response, and decides whether or not to submit a follow-up activity or transaction. For this type of workload, the most important measure of its performance is how quickly the user gets back an individual result. The processing speed of the database manager to process a single activity or transaction for each individual user on the system is the main determinant of how quickly the average user can get back an individual result. How quickly the database manager can process all the activities or transactions from a user over a given period of time is not the relevant metric because it is more dependent on user behavior than on the performance of the database manager.

**Performance measures**

You can use the following performance measures to ascertain how well your system is performing under a particular type of workload.

**Average throughput**

Average throughput is the average number of service completions per unit time. If the service is a transaction or unit of work (UOW), then the average UOW throughput is the number of unit of work completions per unit time. It is usually presented as transactions per second or transactions per minute. Average throughput is a useful measure of system performance when the type of work being measured is a batch workload.

Average activity throughput is the average number of activity completions per unit time. On a system with mostly long-running units of work containing many individual activities, it is easier to measure the progress of the activities within the workload by measuring activity throughput rather than measure the progress of the workload by measuring UOW throughput.

**Average response time**
Average response time is the average amount of time it takes to get a single service completion from the time the service was requested. If the service is a transaction or unit of work (UOW), then the average UOW response time is the amount of time it takes for a UOW to complete from the time it was requested. Average response time is a useful measure of system performance when the type of work being measured is a transactional workload. The closest approximation to average UOW response time is the \texttt{uow\_lifetime\_avg} statistic available from the \texttt{MON\_SAMPLE\_SERVICE\_CLASS\_METRICS} and \texttt{MON\_SAMPLE\_WORKLOAD\_METRICS} table functions, the \texttt{WLM\_GET\_SERVICE\_SUBCLASS\_STATS} and \texttt{WLM\_GET\_WORKLOAD\_STATS} table functions, and the \texttt{event\_scstats} and \texttt{event\_wlstats} event monitor logical data groups reported in the WLM statistics event monitor. A more sophisticated form of UOW lifetime information is available in the \texttt{UowLifetime} histogram, also available in the event monitor.

Average activity response time is the average amount of time it takes to get a single activity to return its result from the time the activity was started. The closest approximation to average activity response time is the \texttt{coord\_act\_lifetime\_avg} statistic available from the \texttt{WLM\_GET\_SERVICE\_SUBCLASS\_STATS} and \texttt{WLM\_GET\_WORKLOAD\_STATS} table functions, and the \texttt{event\_scstats} and \texttt{event\_wlstats} event monitor logical data groups. This number is measured at each member and is reset when a member is deactivated or the \texttt{WLM\_COLLECT\_STATS} procedure is called. The reason it can be an approximation is that for one type of activity, a cursor activity, the activity can return some results before it finishes and relies on the user to finish reading the result set and close the cursor before the activity is considered complete. A more sophisticated form of activity lifetime is available in the \texttt{CoordActLifetime} histogram, also available in the event monitor.

**CPU utilization**

Another metric that is useful when tuning workload management dispatcher, regardless of the type of workload, is the CPU utilization. CPU utilization is the fraction of the time that the CPU resources are busy on the host or LPAR. CPU utilization is the metric that the workload management dispatcher uses to allocate the CPU resources to any one service class. CPU utilization is also the metric that you can use to verify that your workload management dispatcher configuration is working the way you intended. You can measure the CPU utilization over the same intervals as the \texttt{uow\_throughput}, \texttt{uow\_lifetime\_avg}, and \texttt{act\_throughput} monitor elements by using the \texttt{MON\_SAMPLE\_SERVICE\_CLASS\_METRICS} and \texttt{MON\_SAMPLE\_WORKLOAD\_METRICS} table functions, the \texttt{WLM\_GET\_SERVICE\_SUBCLASS\_STATS} and \texttt{WLM\_GET\_WORKLOAD\_STATS} table functions, and the \texttt{event\_scstats} and \texttt{event\_wlstats} event monitor logical data groups collected and reported by the WLM statistics event monitor.

**Note:** If CPU utilization measurements are not as expected for your created service classes, check for the presence of workloads that are running under the default user and maintenance service classes because these workloads were not explicitly assigned to service classes that you created. Forgetting to include workloads running under these default service classes, which each have 1000 hard CPU shares assigned by default when CPU shares were first enabled, can account for CPU utilization measurements that are not as you expected.

The CPU utilization reported through the table functions and event monitors is the CPU resources that are consumed by work executing in only the user and maintenance service classes. Work that is not handled by the dispatcher is not counted towards CPU utilization.
Work that is not handled by the workload management dispatcher includes:
- Work performed by applications or middleware products, other than the DB2 database manager, that perform a portion of their work outside of the DB2 database manager
- Work performed by entities executing in the DB2 system service class
- Work performed by other DB2 instances
- Non-DB2 database manager work performed in fenced mode processes (FMPs) such as fenced stored procedures
- Non-DB2 database manager work performed in trusted routines

To obtain the CPU utilization for these other consumers of CPU resources, one must use operating-system-level (OS-level) monitoring such as that provided with OS workload managers.

CPU velocity

CPU velocity is a statistic that determines whether there is contention for a resource and the degree of such contention. When all access to a resource is mutually exclusive and there are multiple requestors of that resource wanting to access it at the same time, there must be some form of queuing for access, or requestors must be turned away. When a queue is allowed to form, the time taken for a requestor to obtain and then finish using a resource can exceed the time spent simply using the resource. The velocity is the ratio of the time spent simply using the resource to the total time spent both waiting for and using the resource. It is measured on a scale of zero to 100%. When there is a high amount of contention for a resource, velocity sinks towards zero. When there is no contention for a resource, there is no queue time; CPU velocity reaches its maximum value of 100%.

When the workload management dispatcher is enabled, you can measure CPU velocity using the MON_SAMPLE_SERVICE_CLASS_METRICS and MON_SAMPLE_WORKLOAD_METRICS table functions, the WLM_GET_WORKLOAD_STATS and WLM_GET_SERVICE_SUBCLASS_STATS table functions, and the event_scstats and event_wlstats event monitor logical data groups collected and reported by the WLM statistics event monitor. A low CPU velocity value indicates that contention exists for the CPU resources of the host or LPAR and indicates that the workload management dispatcher can be effective in shifting CPU resources towards high-priority service classes and away from low-priority service classes. A high CPU velocity indicates that the workload management dispatcher will have a limited effect on improving workload performance, since every request for CPU resources is already being serviced without any delay.

Default query concurrency management

To ensure that heavier workloads that use column-organized data do not overload the system when many queries are submitted simultaneously, there is a limit on the number of “heavyweight” queries that can run against a database at the same time.

You can implement this limit by using the default workload management concurrency threshold. This threshold is automatically enabled on new databases if you set the value of the DB2_WORKLOAD registry variable to ANALYTICS. You can manually enable the threshold on existing databases.
The processing of queries against column-organized tables is designed to run fast by using the highly parallelized in-memory processing of data. The trade-off for this high performance is that queries that reference column-organized tables also use a relatively large amount of memory and CPU when compared to similar queries processing row-organized table data. As such, the execution of queries that process column-organized tables is optimal when relatively few queries are admitted to the system at a time. When this occurs, the queries can use more processing power and memory, and contention for the processor caches is minimized.

Also, field experience with DB2 workload management has demonstrated that in analytic environments that support mixed workloads, controlling the admission of heavyweight queries into the system yields improvements in both system stability and overall performance. A mixed workload environment is one in which queries might vary widely in their degree of complexity and resource needs. The reason for the improvements is that resource overload on the system is avoided. When the limit on the number of heavyweight queries is reached, the remaining queries are queued and must wait until other queries are complete before they can run. Queuing can help to ensure system stability when many complex ad hoc queries are running on systems that do not have a specific workload management strategy.

To further optimize the execution of mixed workloads on your system, review the full range of DB2 workload management capabilities. For details, see Implementing DB2 workload management in a data warehouse.

**Default workload management objects for concurrency control**

For concurrency control, several default workload management objects are created for new or upgraded databases.

- A service subclass, SYSDEFAULTMANAGEDSUBCLASS, under the SYSDEFAULTUSERCLASS superclass, where heavyweight queries run and can be controlled and monitored as a group.
- A CONCURRENTDBCOORDACTIVITIES threshold, SYSDEFAULTCONCURRENT, which is applied to the SYSDEFAULTMANAGEDSUBCLASS subclass to control the number of concurrently running queries that are running in that subclass.
- A work class set, SYSDEFAULTUSERWCS, and a work class, SYSMANAGEDQUERIES, which identify the class of heavyweight queries to control. The SYSMANAGEDQUERIES work class includes queries that are classified as READ DML (a work type for work classes) and that exceed a timeron threshold that reflects heavier queries.
- A work action set, SYSDEFAULTUSERWAS, and work action, SYSMAPMANAGEDQUERIES, which map all queries that fall into the SYSMANAGEDQUERIES work class to the SYSDEFAULTMANAGEDSUBCLASS service subclass.

When you create or upgrade a database, the following behavior applies to the new default objects:

- The work action set is enabled by default so that queries that meet the criteria that you specify for the SYSMANAGEDQUERIES work class run in the SYSDEFAULTMANAGEDSUBCLASS service subclass.
- If you set the value of the DB2_WORKLOAD registry variable to ANALYTICS, the threshold on the SYSDEFAULTMANAGEDSUBCLASS service subclass is enabled by default for newly created databases only.
• Running the DB2 Configuration Advisor causes the work action set and the
work action to be enabled if you disabled either of them and causes the
threshold to be enabled or disabled, depending on whether you set the
DB2_WORKLOAD registry variable to ANALYTICS.
• You can manually enable or disable the work action set and the threshold. For
more information about manual configuration, see “Adjusting the default
configuration.”

In a customized workload management environment, the expectation is that this
default concurrency control might not be required. You can leave it enabled to
manage the work that maps to the SYSDEFAULTUSERCLASS superclass or disable
it.

The default concurrency limit

A limit on the number of queries that can run concurrently should protect the
system from overload but also avoid over-throttling the workload to a point where
performance might suffer. This objective can be challenging when a system is
running ad hoc analytic queries that range widely in complexity and impact. The
default limit is based on a heuristic calculation that factors in system hardware
attributes such as the number of CPU sockets, CPU cores, and threads per core.
The limit is calculated based on the current host configuration when the default
workload management objects for concurrency control are created during database
creation or upgrade. The limit is recalculated when you run the DB2 Configuration
Advisor against the database. Rerun the DB2 Configuration Advisor if you make
changes to the host, such as enabling or removing CPU capacity.

Adjusting the default configuration

Although the objective is to provide basic query management for analytic data
mart deployments without the need for manual intervention, there might be
scenarios where adjustments to the default configuration are a good idea.

Only the following subset of workload management DDL statements is supported
by the default workload management objects for concurrency control in DB2
Version 10.5.

You can enable or disable the mapping of heavyweight READ DML queries to the
SYSDEFAULTMANAGEDSUBCLASS subclass, as shown in the following
examples:

# Enable the default work action mapping so that
# heavyweight queries are mapped to the
# SYSDEFAULTMANAGEDSUBCLASS subclass
ALTER WORK ACTION SET SYSDEFAULTUSERWAS ENABLE

# Disable the default work action mapping so that all
# queries are mapped to the SYSDEFAULTSUBCLASS subclass
ALTER WORK ACTION SET SYSDEFAULTUSERWAS DISABLE

You can adjust the timeron range for the mapping of heavyweight READ DML
queries, as shown in the following example:

# Modify the default work class timeron range from A to B
ALTER WORK CLASS SET SYSDEFAULTUSERWCS
  ALTER WORK CLASS SYSMANAGEDQUERIES FOR TIMERONCOST FROM <A> TO <B>

You can enable or disable the concurrency threshold on the
SYSDEFAULTMANAGEDSUBCLASS subclass, as shown in the following example:
# Enable the concurrency threshold
ALTER THRESHOLD SYSDEFAULTCONCURRENT ENABLE

# Disable the concurrency threshold
ALTER THRESHOLD SYSDEFAULTCONCURRENT DISABLE

You can adjust the concurrency limit, as shown in the following example:

# Change the concurrency limit to [N]
ALTER THRESHOLD SYSDEFAULTCONCURRENT
    WHEN CONCURRENTDBCOORDACTIVITIES > [N] STOP EXECUTION

The following optional clauses are supported when you alter the SYSDEFAULTCONCURRENT threshold:

alter-threshold-exceeded-actions
AND QUEUEDACTIVITIES

## Tuning the settings

The recommended method for tuning the default object settings is to examine the overall system resource usage with a full workload and to perform incremental adjustments that are based on whether the system resources appear to be under-utilized or over-utilized. A system with a run queue depth of approximately 10 times the number of CPU cores and physical memory usage under 100% is considered to be in a fully used and healthy state.

The following recommendations represent a starting point for tuning the default concurrency threshold value and the default work class timeron range when the system appears to be under-utilized or over-utilized. This guidance applies to a non-customized workload management environment. For a more comprehensive set of recommendations that applies to monitoring both system use and workload characteristics, see Implementing DB2 workload management in a data warehouse.

- If the system appears to be under-utilized, take the following steps:
  1. Examine the WLM_QUEUE_TIME_TOTAL metric, which is reported by various system-level table functions and the statistics event monitor, to determine whether queries in the system are accumulating time by waiting on concurrency thresholds.
     a. If no such queue time is being accumulated, the system is running under peak capacity, and no tuning is necessary.
     b. If queue time is being accumulated, take the following steps:
        1) Monitor the amount of work that is running in the SYSDEFAULTSUBCLASS and SYSDEFAULTMANAGEDSUBCLASS classes.
        2) If it appears that too large a proportion of the workload is running within the SYSMANAGEDQUERIES class, consider incrementally increasing the TIMERONCOST minimum for the SYSMANAGEDQUERIES class.
  2. If the distribution of managed and unmanaged work appears reasonable, consider incrementally increasing the concurrency limit that is specified by the SYSDEFAULTCONCURRENT threshold until system resource usage reaches the target level.

- If the system appears to be over-utilized, take the following steps:
  1. Monitor the amount of work that is running in the SYSDEFAULTSUBCLASS and SYSDEFAULTMANAGEDSUBCLASS classes.
2. If it appears that too small a proportion of the workload is running within the SYSMANAGEDQUERIES class, consider incrementally decreasing the TIMERONCOST minimum for the SYSMANAGEDQUERIES class.

3. If the distribution of managed and unmanaged work appears reasonable, consider incrementally decreasing the concurrency limit that is specified by the SYSDEFAULTCONCURRENT threshold until system resource usage is back within the target range.

**Default workload management objects for concurrency control**

For concurrency control, several default workload management objects are created for new or upgraded databases.

- **SYSDEFAULTMANAGEDSUBCLASS**, a service subclass under the SYSDEFAULTUSERCLASS service superclass, where heavyweight queries run and can be controlled and monitored as a group
- **SYSDEFAULTCONCURRENT**, a CONCURRENTDBCOORDACTIVITIES threshold that is applied to the SYSDEFAULTMANAGEDSUBCLASS subclass to control the number of concurrently running queries that are running in that subclass
- **SYSDEFAULTUSERWCS** (a work class set) and **SYSMANAGEDQUERIES** (a work class), which identify the class of heavyweight queries to control
- **SYSDEFAULTUSERWAS** (a work action set) and **SYMAPMANAGEDQUERIES** (a work action), which map all queries in the SYSMANAGEDQUERIES work class to the SYSDEFAULTMANAGEDSUBCLASS service subclass

**Default managed subclass (SYSDEFAULTMANAGEDSUBCLASS)**

You can query the SYSCAT.SERVICECLASSES catalog view to obtain information about the SYSDEFAULTMANAGEDSUBCLASS service subclass. The following table shows the columns, their values, and your ability to modify these values. For information about how to enable or disable the mapping of heavyweight READ DML queries to the SYSDEFAULTMANAGEDSUBCLASS service subclass, see “Default query concurrency management” on page 213.

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER SERVICE CLASS statement if you have DBADM or WLMADM authority and, for COLLECT clauses, SQLADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICECLASSNAME</td>
<td>SYSDEFAULTMANAGEDSUBCLASS</td>
<td>No</td>
</tr>
<tr>
<td>PARENTSERVICECLASSNAME</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>No</td>
</tr>
<tr>
<td>SERVICECLASSID</td>
<td>4</td>
<td>No</td>
</tr>
<tr>
<td>PARENTID</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>Time stamp of database creation or upgrade</td>
<td>No</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>Time stamp of the last update of this object</td>
<td>No</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Y</td>
<td>No</td>
</tr>
<tr>
<td>AGENTPRIORITY</td>
<td>-32768 (default)</td>
<td>Yes</td>
</tr>
<tr>
<td>PREFETCHPRIORITY</td>
<td>’ ’ (default)</td>
<td>Yes</td>
</tr>
<tr>
<td>MAXDEGREE</td>
<td>NULL</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 47. SYSDEFAULTMANAGEDSUBCLASS entry in SYSCAT.SERVICECLASSES view (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER SERVICE CLASS statement if you have DBADM or WLMADM authority and, for COLLECT clauses, SQLADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERPOOLPRIORITY</td>
<td>‘ ’ (default)</td>
<td>Yes</td>
</tr>
<tr>
<td>INBOUNDCORRELATOR</td>
<td>NULL</td>
<td>No</td>
</tr>
<tr>
<td>OUTBOUNDCORRELATOR</td>
<td>NULL</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTAGGACTDATA</td>
<td>N (none)</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTAGGREQDATA</td>
<td>N (none)</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTACTDATA</td>
<td>N (none)</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTACTPARTITION</td>
<td>C (coordinator member of the activity)</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTREQMETRICS</td>
<td>N (none)</td>
<td>Yes</td>
</tr>
<tr>
<td>CPUSHARES</td>
<td>1000</td>
<td>Yes</td>
</tr>
<tr>
<td>CPUSHARETYPE</td>
<td>H (hard shares)</td>
<td>Yes</td>
</tr>
<tr>
<td>CPULIMIT</td>
<td>-1 (no CPU limit)</td>
<td>Yes</td>
</tr>
<tr>
<td>SORTMEMORYPRIORITY</td>
<td>NULL</td>
<td>No</td>
</tr>
<tr>
<td>SECTIONACTUALSOPTIONS</td>
<td>NN (not enabled)</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTAGGUOWDATA</td>
<td>N (none)</td>
<td>Yes</td>
</tr>
<tr>
<td>REMARKS</td>
<td>NULL</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For more information, see SYSCAT.SERVICECLASSES.

**Default user work class set (SYSDEFAULTUSERWCS)**

You can query the SYSCAT.WORKCLASSSETS catalog view to obtain information about the SYSDEFAULTUSERWCS work class set. The following table shows the columns, their values, and your ability to modify these values. For more information, see “Default query concurrency management” on page 213.

Table 48. SYSDEFAULTUSERWCS entry in SYSCAT.WORKCLASSSETS view

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER WORK CLASS SET statement if you have DBADM or WLMADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKCLASSSETNAME</td>
<td>SYSDEFAULTUSERWCS</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>Time stamp of database creation or upgrade</td>
<td>No</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>Time stamp of the last update of this object</td>
<td>No</td>
</tr>
<tr>
<td>REMARKS</td>
<td>NULL</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For more information, see SYSCAT.WORKCLASSSETS.
Managed queries work class (SYSMANAGEDQUERIES)

You can query the SYSCAT.WORKCLASSES catalog view to obtain information about the SYSMANAGEDQUERIES work class. The following table shows the columns, their values, and your ability to modify these values. For more information, see “Default query concurrency management” on page 213.

Table 49. SYSMANAGEDQUERIES entry in SYSCAT.WORKCLASSES view

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER WORK CLASS SET statement if you have DBADM or WLMADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKCLASSNAME</td>
<td>SYSMANAGEDQUERIES</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETNAME</td>
<td>SYSDEFAULTUSERWCS</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>Time stamp of database creation or upgrade</td>
<td>No</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>Time stamp of the last update of this object</td>
<td>No</td>
</tr>
<tr>
<td>EVALUATIONORDER</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>

For more information, see SYSCAT.WORKCLASSES.

You can also query the SYSCAT.WORKCLASSATTRIBUTES catalog view to obtain information about the SYSMANAGEDQUERIES work class. The following tables show the columns, their values, and your ability to modify these values.

Table 50. SYSMANAGEDQUERIES entries in SYSCAT.WORKCLASSATTRIBUTES view. Attribute 1

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER WORK CLASS SET statement if you have DBADM or WLMADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKCLASSNAME</td>
<td>SYSMANAGEDQUERIES</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETNAME</td>
<td>SYSDEFAULTUSERWCS</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>TYPE</td>
<td>1 (work type attribute)</td>
<td>No</td>
</tr>
<tr>
<td>VALUE1</td>
<td>2 (read activities)</td>
<td>No</td>
</tr>
<tr>
<td>VALUE2</td>
<td>NULL</td>
<td>No</td>
</tr>
<tr>
<td>VALUE3</td>
<td>NULL</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 51. SYSMANAGEDQUERIES entries in SYSCAT.WORKCLASSATTRIBUTES view. Attribute 2

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER WORK CLASS SET statement if you have DBADM or WLMADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKCLASSNAME</td>
<td>SYSMANAGEDQUERIES</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETNAME</td>
<td>SYSDEFAULTUSERWCS</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 51. SYSMANAGEDQUERIES entries in SYSCAT.WORKCLASSATTRIBUTES view (continued). Attribute 2

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER WORK CLASS SET statement if you have DBADM or WLMADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKCLASSID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>TYPE</td>
<td>2 (timeron cost attribute)</td>
<td>No</td>
</tr>
<tr>
<td>VALUE1</td>
<td>Minimum timeron cost, computed at database creation or when the DB2 Configuration Advisor is run</td>
<td>Yes</td>
</tr>
<tr>
<td>VALUE2</td>
<td>-1 (maximum timeron cost; unbounded)</td>
<td>Yes</td>
</tr>
<tr>
<td>VALUE3</td>
<td>NULL</td>
<td>No</td>
</tr>
</tbody>
</table>

For more information, see SYSCAT.WORKCLASSATTRIBUTES.

**Default user work action set (SYSDEFAULTUSERWAS)**

You can query the SYSCAT.WORKACTIONSETS catalog view to obtain information about the SYSDEFAULTUSERWAS work action set. The following table shows the columns, their values, and your ability to modify these values. For information about how to enable or disable the mapping of heavyweight READ DML queries to the SYSDEFAULTMANAGEDSUBCLASS service subclass, see “Default query concurrency management” on page 213.

Table 52. SYSDEFAULTUSERWAS entry in SYSCAT.WORKACTIONSETS view

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER WORK ACTION SET statement if you have DBADM or WLMADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONSETNAME</td>
<td>SYSDEFAULTUSERWAS</td>
<td>No</td>
</tr>
<tr>
<td>ACTIONSETID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETNAME</td>
<td>SYSDEFAULTUSERWCS</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>Time stamp of database creation or upgrade</td>
<td>No</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>Time stamp of the last update of this object</td>
<td>No</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>OBJECTTYPE</td>
<td>b (work action set applies to service superclass)</td>
<td>No</td>
</tr>
<tr>
<td>OBJECTNAME</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>No</td>
</tr>
<tr>
<td>OBJECTID</td>
<td>3 (service class ID of SYSDEFAULTUSERCLASS)</td>
<td>No</td>
</tr>
<tr>
<td>REMARKS</td>
<td>NULL</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For more information, see SYSCAT.WORKACTIONSETS.
Managed queries work action (SYSMAPMANAGEDQUERIES)

You can query the SYSCAT.WORKACTIONS catalog view to obtain information about the SYSMAPMANAGEDQUERIES work action. The following table shows the columns, their values, and your ability to modify these values. For more information, see “Default query concurrency management” on page 213.

Table 53. SYSMAPMANAGEDQUERIES entry in SYSCAT.WORKACTIONS view

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER WORK ACTION SET statement if you have DBADM or WLMADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONNAME</td>
<td>SYSMAPMANAGEDQUERIES</td>
<td>No</td>
</tr>
<tr>
<td>ACTIONID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>ACTIONSETNAME</td>
<td>SYSDEFAULTUSERWAS</td>
<td>No</td>
</tr>
<tr>
<td>ACTIONSETID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSNAME</td>
<td>SYSMANAGEDQUERIES</td>
<td>No</td>
</tr>
<tr>
<td>WORKCLASSID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>Time stamp of database creation or upgrade</td>
<td>No</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>Time stamp of the last update of this object</td>
<td>No</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>ACTIONTYPE</td>
<td>M (map activities to service subclass)</td>
<td>No</td>
</tr>
<tr>
<td>REFOBJECTID</td>
<td>4 (service class ID for SYSDEFAULTMANAGEDSUBCLASS)</td>
<td>No</td>
</tr>
<tr>
<td>REFOBJECTTYPE</td>
<td>NULL</td>
<td>No</td>
</tr>
<tr>
<td>SECTIONACTUALSOPTIONS</td>
<td>NN (not enabled)</td>
<td>No</td>
</tr>
</tbody>
</table>

For more information, see SYSCAT.WORKACTIONS.

Default concurrent threshold (SYSDEFAULTCONCURRENT)

You can query the SYSCAT.THRESHOLDS catalog view to obtain information about the SYSDEFAULTCONCURRENT threshold. The following table shows the columns, their values, and your ability to modify these values. For more information, see “Default query concurrency management” on page 213.

Table 54. SYSDEFAULTCONCURRENT entry in SYSCAT.THRESHOLDS view

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Modifiable by using the ALTER THRESHOLD statement if you have DBADM or WLMADM authority and, for COLLECT clauses, SQLADM authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRESHOLDNAME</td>
<td>SYSDEFAULTCONCURRENT</td>
<td>No</td>
</tr>
<tr>
<td>THRESHOLDID</td>
<td>2147483647</td>
<td>No</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>U (not created by work action)</td>
<td>No</td>
</tr>
<tr>
<td>THRESHOLDCLASS</td>
<td>A (aggregate)</td>
<td>No</td>
</tr>
<tr>
<td>THRESHOLDPREDICATE</td>
<td>CONCDBC (concurrent database coordinator activities)</td>
<td>No</td>
</tr>
<tr>
<td>Column</td>
<td>Value</td>
<td>Modifiable by using the ALTER THRESHOLD statement if you have DBADM or WLMADM authority and, for COLLECT clauses, SQLADM authority</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>THRESHOLDPREDICATEID</td>
<td>90</td>
<td>No</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>SB (service subclass)</td>
<td>No</td>
</tr>
<tr>
<td>DOMAINID</td>
<td>4 (service class ID of SYSDEFAULTMANAGEDSUBCLASS)</td>
<td>No</td>
</tr>
<tr>
<td>ENFORCEMENT</td>
<td>D (database)</td>
<td>No</td>
</tr>
<tr>
<td>QUEUING</td>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>MAXVALUE</td>
<td>Computed at database creation or when the DB2 Configuration Advisor is run, based on the host hardware configuration</td>
<td>Yes</td>
</tr>
<tr>
<td>DATATAGLIST</td>
<td>NULL</td>
<td>No</td>
</tr>
<tr>
<td>QUEUESIZE</td>
<td>-1 (unbounded)</td>
<td>Yes</td>
</tr>
<tr>
<td>OVERFLOWPERCENT</td>
<td>-1</td>
<td>No</td>
</tr>
<tr>
<td>COLLECTACTDATA</td>
<td>N (none)</td>
<td>Yes</td>
</tr>
<tr>
<td>COLLECTACTPARTITION</td>
<td>C (coordinator member of the activity)</td>
<td>Yes</td>
</tr>
<tr>
<td>EXECUTION</td>
<td>S (stop execution when queue is full)</td>
<td>Yes</td>
</tr>
<tr>
<td>REMAPSCID</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>VIOLATIONRECORDLOGGED</td>
<td>NULL</td>
<td>No</td>
</tr>
<tr>
<td>CHECKINTERVAL</td>
<td>-1</td>
<td>No</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>Time stamp of database creation or upgrade</td>
<td>No</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>Time stamp of the last update of this object</td>
<td>No</td>
</tr>
<tr>
<td>REMARKS</td>
<td>NULL</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For more information, see SYSCAT.THRESHOLDS.

**Canceling activities**

If an activity is consuming too many resources, or is running too long, you can cancel it. Canceling an activity is gentler than forcing the application that submitted the activity. A canceled activity returns SQL4725N to the user, but does not end the connection or affect any other user activity. Forcing the application ends both the connection and user activities.

**About this task**

You can only explicitly cancel an activity if a coordinator activity is currently working on a request for the activity. If you cancel an activity in the IDLE state (that is, no requests are being processed), the activity is placed in the CANCEL_PENDING state and is canceled on the next request that is received. For
example, if you attempt to cancel a CURSOR activity between fetches, the SQL4725N error is not returned to the user until the next fetch after the cancel.

All user activities are cancelable, including the load utility and stored procedures.

**Procedure**

1. Identify the activity that you want to cancel. You can use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function to identify the activities running in an application. You can also use the MON_GET_ACTIVITY_DETAILS_COMPLETE table function to view additional details about a particular activity if the information in WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES is not sufficient to identify the work that the activities are performing.

2. Cancel the activity using the WLMCANCEL_ACTIVITY stored procedure. The stored procedure takes the following arguments: application_handle, uow_id, and activity_id. For an example of how to use this stored procedure, see “Scenario: Identifying activities that are taking too long to complete” on page 352.
Chapter 4. Monitoring and intervention

The third domain of workload management is monitoring, which must be performed on an ongoing basis.

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

Looking at aggregate information should usually be the first step when you build a monitoring strategy. Aggregates give a good picture of overall data server activity and are also cheaper because you do not have to collect information on every activity in which you might be interested. You can collect more detailed information as you understand the scope of your monitoring needs.

Typical monitoring tasks you can perform are:

- Analyzing the workload on your system to help design your initial DB2 workload management configuration.
- Tracking and investigating the behavior of your system by obtaining types of operational information that permit you to:
  - Analyze system performance degradation
  - Diagnose activities that are taking too long to complete
  - Investigate agent contention
  - Isolate poorly performing queries

Information is available for activities, service classes, workloads, work classes, threshold queues, and threshold violations.

- Exercising control over the execution environment by canceling queued activities that you expect will cause problems or cancel running activities that you have diagnosed as negatively impacting the system.

Real-time monitoring with table functions

Real-time monitoring data includes information about work currently running on the system, statistics, and metrics for work that has been performed on the system that can help you to determine usage patterns and resource allocation and identify problem areas. You use DB2 table functions to obtain this operational information.

Table functions with names that begin with \texttt{WLM} are DB2 workload management table functions. These table functions provide access to a set of data relevant to managing your workload, such as workload management statistics, as a virtual DB2 table against which you can issue a SELECT statement. This enables you to write applications to query data and analyze it as if it were in a physical table on the data server. The DB2 workload management table functions are qualified with the \texttt{SYSPROC} schema name.

Table functions with names that begin with \texttt{MON} are monitoring metrics functions. Monitoring metrics provide monitoring data about the health of and query performance on your DB2 data server, which can then be used as input to a
3rd party tool or in conjunction with additional scripting you provide to analyze 
the metrics returned. Only those monitoring metrics functions that are relevant for 
DB2 workload management are included here. The monitor metrics table functions 
are similar to the workload management statistics table functions. Both return 
elements describing work that has taken place on the system. The key differences 
between these monitoring metrics table functions and the DB2 workload 
management table functions are:

- The DB2 workload management table functions provide data that is more 
  statistical in nature, such as computed values like averages, high watermarks, 
  standard deviations, etc. In contrast, the monitoring metrics table functions 
  provide a much more complete set of raw monitoring data.

- The data reported by the DB2 statistics functions is reset when data is sent to a 
  statistics event monitor. This resetting of data is necessary to make values such 
  as high watermarks meaningful over a specific collection interval. Data reported 
  by the monitoring metrics functions is also captured by a statistics event 
  monitor, but is never reset. The data reported by monitoring interfaces 
  accumulates from the time a database is activated until the time it is deactivated.

Some table functions return sets of information about the work that is currently 
running on a system:

Table 55. Table functions that show you the work currently running on the system

<table>
<thead>
<tr>
<th>Objects for which information is collected</th>
<th>Functions and information returned</th>
</tr>
</thead>
</table>
| Workload occurrences                      | The WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function returns a list of workload occurrences, across database members, that are assigned to a service class. For each occurrence, there is information about the current state and the connection attributes used to assign the workload to the service class and activity statistics indicating activity volume and success rates. For an example of how to use this table function, see “Example: Investigating agent usage by service class” on page 90.

The deprecated WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 table function is also available.

| Workload occurrence activities | The WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function returns a list of current activities associated with a workload occurrence. For each activity, information is available about the current state of the activity (for example, executing or queued), the type of activity (for example, LOAD, READ, or DDL), and the time at which the activity started. For examples of how to use this table function, see “Example: Aggregating data using DB2 workload management table functions” on page 235 and “Scenario: Identifying activities that are taking too long to complete” on page 352.

The deprecated WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 table function is also available. |
### Table 55. Table functions that show you the work currently running on the system (continued)

<table>
<thead>
<tr>
<th>Objects for which information is collected</th>
<th>Functions and information returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service class agents</td>
<td>The WLM_GET_SERVICE_CLASS_AGENTS table function returns a list of database agents associated with a service class or an application handle. Information returned also shows the current state of the agent, the action that the agent is performing, and the status of that action. For an example of how to use this table function, see “Example: Investigating agent usage by service class” on page 90. The deprecated WLM_GET_SERVICE_CLASS_AGENTS_V97 table function is also available.</td>
</tr>
<tr>
<td>Activities</td>
<td>The MON_GET_ACTIVITY_DETAILS table function returns metrics and other detailed information as an XML document about a specific activity identified by its application handle, unit of work ID, and activity ID. One detail returned is the activity type; depending on that type, a set of additional data is returned. For example, for SQL activities, cost estimates and information about the statement text, package data, and rows returned or modified are provided. Details about the isolation level and processor resource are also available, amongst others. The deprecated WLM_GET_ACTIVITY_DETAILS table function is also available. For an example that shows you how to use this table function, see “Example: Monitoring current system behavior at different levels using DB2 workload management table functions” on page 231.</td>
</tr>
</tbody>
</table>

Some table functions return monitoring data for all requests executed on the system aggregated by service subclass and workload objects:

### Table 56. Table functions that show you monitoring data aggregated by DB2 workload management objects

<table>
<thead>
<tr>
<th>Objects for which data is aggregated</th>
<th>Functions and information returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workloads</td>
<td>Both the MON_GET_WORKLOAD table function and the MON_GET_WORKLOADDETAILS table function return metrics for one or more workloads. The metrics returned by this function represent the accumulation of all metrics of all workload occurrences that use the same workload definition. The MON_GET_WORKLOAD table function returns the most commonly used metrics in a column-based format and is an efficient method of retrieving base metrics. The MON_GET_WORKLOADDETAILS table function returns the entire set of available metrics in an XML document format, which provides maximum flexibility for formatting output. The XML based output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function.</td>
</tr>
</tbody>
</table>
Table 56. Table functions that show you monitoring data aggregated by DB2 workload management objects (continued)

<table>
<thead>
<tr>
<th>Objects for which data is aggregated</th>
<th>Functions and information returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service subclasses</td>
<td>Both the MON_GET_SERVICE_SUBCLASS table function and the MON_GET_SERVICE_SUBCLASS_DETAILS table function return metrics for one or more service subclasses. The metrics returned by the table functions represent the accumulation of all metrics for requests that have executed under the indicated service subclass. The MON_GET_SERVICE_SUBCLASS table function returns the most commonly used metrics in a column based format and is an efficient method of retrieving base metrics. The MON_GET_SERVICE_SUBCLASS_DETAILS table function returns the entire set of available metrics in an XML document format, which provides maximum flexibility for formatting output. The XML-based output can be parsed directly by an XML parser, or it can be converted to relational format by the XMLTABLE function.</td>
</tr>
<tr>
<td>Connections</td>
<td>The MON_GET_CONNECTION table function returns data that is aggregated across user connections to the system.</td>
</tr>
<tr>
<td>Units of work</td>
<td>The MON_GET_UNIT_OF_WORK table function returns data that is aggregated for the current unit of work within a user connection.</td>
</tr>
</tbody>
</table>

**Statistical information**

General statistical information is also available for a number of different objects. You can use this statistical information for a number of different purposes, such as for verifying that changes to your DB2 workload management configuration have had the expected effect. If you create a new work class to classify READ activities, for example, you can verify that READ activities are being classified under the new work class correctly. You can also use table functions to quickly recognize certain problems with the system. For example, you can use table functions to determine an acceptable value for the average activity lifetime and recognize when this value exceeds its usual range, possibly indicating a problem that requires further investigation.

The following table lists the statistics that you can obtain by using table functions. All statistics table functions return the statistics that accumulated since the last time that you reset the statistics.

Table 57. Table functions that show you statistical information

<table>
<thead>
<tr>
<th>Objects for which statistics are returned</th>
<th>Functions and statistics returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service superclasses</td>
<td>The WLM_GET_SERVICE_SUPERCLASS_STATS table function shows summary statistics across database members at the service superclass level: namely, high-water marks for concurrent connections, which are useful when determining peak workload activity.</td>
</tr>
</tbody>
</table>
Table 57. Table functions that show you statistical information (continued)

<table>
<thead>
<tr>
<th>Objects for which statistics are returned</th>
<th>Functions and statistics returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service subclasses</td>
<td>The WLM_GET_SERVICE_SUBCLASS_STATS table function shows summary statistics across database members at the service subclass level (all activities run in service subclasses). Statistics include numbers of completed activities and average execution times. This information is useful when you are looking at general system health and distribution of activities across service classes and database members. For examples of how to use this table function, see “Example: Obtaining point-in-time statistics from service classes” on page 234, “Example: Aggregating data using DB2 workload management table functions” on page 235, “Example: Analyzing a service class-related system slowdown” on page 89, and “Scenario: Investigating a workload-related system slowdown” on page 348. The deprecated WLM_GET_SERVICE_SUBCLASS_STATS_V97 table function is also available.</td>
</tr>
<tr>
<td>Workloads</td>
<td>The WLM_GET_WORKLOAD_STATS table function shows summary statistics across database members at the workload level. These include high-water marks for concurrent workload occurrences and numbers of completed activities. This information is useful when you are monitoring general system health or drilling down to identify problem areas. For an example of how to use this table function, see “Scenario: Investigating a workload-related system slowdown” on page 348. The deprecated WLM_GET_WORKLOAD_STATS_V97 table function is also available.</td>
</tr>
<tr>
<td>Work action sets</td>
<td>The WLM_GET_WORK_ACTION_SET_STATS table function shows summary statistics across database members at the work action set level: namely, the number of activities in each work class that had the corresponding work actions applied to them. This information is useful for understanding the effectiveness of a work action set and understanding the types of activities running on the system. For an example of how to use this table function, see “Example: Analyzing workloads by activity type” on page 61.</td>
</tr>
<tr>
<td>Threshold queues</td>
<td>The WLM_GET_QUEUE_STATS table function shows summary statistics across database members for the queues used for thresholds. Statistics include the current and total numbers of queued activities and total time spent in a queue. This information is useful when you are querying current queued activity or validating that you defined a threshold correctly. Excessive queuing might indicate that a threshold is too restrictive, and very little queuing might indicate that a threshold is not restrictive enough or not needed.</td>
</tr>
</tbody>
</table>

Statistics are useful only if the time period during which they are collected is meaningful. Collecting statistics over a very long time, and for any length of time using the WLM_COLLECT_STATS stored procedure, might be less useful if it becomes difficult to identify changes to trends or problem areas because there is too much old data. Thus, you can reset statistics at any time.

Because of the default workload and default user service classes, monitoring capabilities exist from the moment that you install the DB2 data server. These can help you to start identifying sources of activities that you can use to create
workloads and the service classes to which you can assign them.

Example: Using DB2 workload management table functions

A large amount of data is available through DB2 workload management real-time monitoring. The example in this topic shows how you might start using the information.

In this situation, only the default workload and service class are in place. Use this example to understand how you can use the table functions to understand what, exactly, is running on the data server. Follow these steps:

1. Use the Service Superclass Statistics table function to show all of the service superclasses. After you install or upgrade to DB2 9.5 or later, three default superclasses are defined: one for maintenance activities, one for system activities, and one for user activities. SYSDEFAULTUSERCLASS is the service class of interest.

   ```sql
   SELECT VARCHAR(SERVICE_SUPERCLASS_NAME,30) AS SUPERCLASS
   FROM TABLE(WLM_GET_SERVICE_SUPERCLASS_STATS('',-1)) AS T
   ```

<table>
<thead>
<tr>
<th>SUPERCLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTSYSTEMCLASS</td>
</tr>
<tr>
<td>SYSDEFAULTMAINTENANCECLASS</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
   3 record(s) selected.

2. Use the Service Subclass Statistics table function to show statistics for all the service subclasses of the SYSDEFAULTUSERCLASS superclass. For each service subclass you can see the current volume of requests that are being processed, the number of activities that have completed execution, and the overall distribution of activities across members (possibly indicating a problem if the distribution is uneven). You can optionally obtain additional statistics including the average lifetime for activities, the average amount of time activities spend queued, and so on. You can obtain optional statistics for a service subclass by specifying the COLLECT AGGREGATE ACTIVITY DATA keyword on the ALTER SERVICE CLASS statement to enable aggregate activity statistics collection.

   ```sql
   SELECT VARCHAR(SERVICE_SUPERCLASS_NAME, 20) AS SUPERCLASS,
          VARCHAR(SERVICE_SUBCLASS_NAME, 20) AS SUBCLASS,
          COORD_ACT_COMPLETED_TOTAL,
          COORD_ACT_ABORTED_TOTAL,
          COORD_ACT_REJECTED_TOTAL,
          CONCURRENT_ACT_TOP
   FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('SYSDEFAULTUSERCLASS', 'SYSDEFAULTSUBCLASS', -1)) AS T
   ```

<table>
<thead>
<tr>
<th>SUPERCLASS</th>
<th>SUBCLASS</th>
<th>COORD_ACT_COMPLETED_TOTAL</th>
<th>COORD_ACT_ABORTED_TOTAL</th>
<th>COORD_ACT_REJECTED_TOTAL</th>
<th>CONCURRENT_ACT_TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
   1 record(s) selected.

3. For a given service subclass, use the Workload Occurrence Information table function to list the occurrences of a workload that are mapped to the service subclass. The table function displays all of the connection attributes, which you can use to identify the source of the activities. This information can be quite useful in determining custom workload definitions in the future. For example, perhaps a specific workload occurrence listed here has a large volume of work from an application as shown by the activities completed counter.
a. For that application, use the Workload Occurrence Activities Information table function to show the current activities across database members that were created from the application's connection. You can use this information for a number of purposes, including identifying activities that might be causing problems on the data server.

```
SELECT APPLICATION_HANDLE, 
    LOCAL_START_TIME, 
    UOW_ID, 
    ACTIVITY_ID, 
    ACTIVITY_TYPE 
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(431,-1)) AS T 
APPLICATION_HANDLE LOCAL_START_TIME UOW_ID ACTIVITY_ID ACTIVITY_TYPE 
------------------------------------------------------------------------ 
431 2008-06-17-12.49.46.854259 11 1 READ_DML
```
1 record(s) selected.

b. For each activity, retrieve more detailed information by using the Activity Details table function. The data might show that some SQL statements are returning huge numbers of rows, that some activities have been idle for a long time, or that some queries are running that have an extremely large estimated cost. In situations such as these, it might make sense to define some thresholds to identify and prevent potentially damaging behavior in the future.

```
SELECT VARCHAR(NAME, 20) AS NAME, 
    VARCHAR(VALUE, 40) AS VALUE 
FROM TABLE(WLM_GET_ACTIVITY_DETAILS(431,11,1,-1)) AS T 
WHERE NAME IN ('UOW_ID', 'ACTIVITY_ID', 'STMT_TEXT') 
NAME VALUE 
----------------------- ---------------------------------------- 
UOW_ID 1 
ACTIVITY_ID 1 
STMT_TEXT select * from syscat.tables
```
3 record(s) selected.

Example: Monitoring current system behavior at different levels using DB2 workload management table functions

DB2 workload management provides a number of table functions that you can use to obtain data about your workload management configuration.

Installing DB2 Version 9.5 or later creates a set of default workloads and service classes. Before deciding how to implement your own DB2 workload management solution, you can use the table functions to observe work being performed in the system in terms of the default workload occurrences, service classes, and activities.

You can start by obtaining the list of workload occurrences in a service class. To do this, use the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table
function. In the following example, an empty string is passed for
*service_superclass_name* and *service_subclass_name*, and -2 (a wildcard character) is
passed for *member*:

```sql
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
      SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
      SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
      SUBSTR(CHARCOORD_MEMBER),1,4) AS COORDMEMB,
      SUBSTR(CHARAPPLICATION_HANDLE),1,7) AS APPHNDL,
      SUBSTR(CHARWORKLOAD_NAME),1,22) AS WORKLOAD_NAME,
      SUBSTR(CHARWORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', -2)) AS SCINFO
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, MEMB, APPHNDL, WORKLOAD_NAME, WLO_ID
```

Assume that the system has four database members and that there are two
applications performing activities on the database when you issue the query. The
results would resemble the following ones:

<table>
<thead>
<tr>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
<th>MEMB</th>
<th>COORDMEMB</th>
<th>APPHNDL</th>
<th>WORKLOAD_NAME</th>
<th>WLO_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>1</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>2</td>
</tr>
</tbody>
</table>

The results indicate that both workload occurrences were assigned to the
SYSDEFAULTUSERWORKLOAD workload. The results also show that both
workload occurrences were assigned to the SYSDEFAULTSUBCLASS service
subclass in the SYSDEFAULTUSERCLASS service superclass and that both
workload occurrences are from the same coordinator member (member 0).

Next, you can also use the
*WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES* table function again
to determine the connection attributes of the two workload occurrences:

```sql
SELECT SUBSTR(CHARAPPLICATION_HANDLE),1,7) AS APPHNDL,
      SUBSTR(CHARWORKLOAD_NAME),1,22) AS WORKLOAD_NAME,
      SUBSTR(CHARWORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID,
      SUBSTR(CHARSYSTEM_AUTH_ID),1,9) AS SYSAUTHID,
      SUBSTR(CHARAPPLICATION_NAME),1,15) AS APPLNAME
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', 0)) AS SCINFO
ORDER BY APPHNDL, WORKLOAD_NAME, WLO_ID
```

<table>
<thead>
<tr>
<th>APPHNDL</th>
<th>WORKLOAD_NAME</th>
<th>WLO_ID</th>
<th>SYSAUTHID</th>
<th>APPLNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>1</td>
<td>LYNN</td>
<td>accountspay</td>
</tr>
<tr>
<td>2</td>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>2</td>
<td>KATE</td>
<td>businessobjects</td>
</tr>
</tbody>
</table>

Then, you can use the *WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES*
table function to show the current activities of one of the workload occurrences:

```sql
SELECT SUBSTR(CHARCOORD_MEMBER),1,5) AS COORD,
      SUBSTR(CHARMEMBER),1,4) AS MEMB,
      SUBSTRCHARUOW_ID),1,5) AS UOWID,
      SUBSTRCHARACTIVITY_ID),1,5) AS ACTID,
      SUBSTRCHARPARENTUOW_ID),1,8) AS PARUOWID,
      SUBSTRCHARPARENT_ACTIVITY_ID),1,8) AS PARACTID,
      SUBSTRACTIVITYTYPE,1,9) AS ACTTYPE,
      SUBSTRCHARNESTINGLEVEL),1,7) AS NESTING
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2)) AS WLOACTS
ORDER BY MEMB, UOWID, ACTID
```

<table>
<thead>
<tr>
<th>COORD</th>
<th>MEMB</th>
<th>UOWID</th>
<th>ACTID</th>
<th>PARUOWID</th>
<th>PARACTID</th>
<th>ACTTYPE</th>
<th>NESTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>CALL</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>READ_DML</td>
<td>1</td>
</tr>
</tbody>
</table>
The query results show that workload occurrence 1 is running two activities. One activity is a stored procedure (indicated by the activity type of CALL), and the other activity is a DML activity that performs a read (for example, a SELECT statement). The DML activity is nested in the stored procedure call. You can tell that the DML activity is nested because the parent unit of work identifier and parent activity identifier of the DML activity match the unit of work identifier and the activity identifier of the CALL activity. You can also tell that the DML activity is executing on database members 0, 1, 2, and 3. The parent identifier information is available only on the coordinator member.

You can obtain more information about an individual activity that is currently running by using the MON_GET_ACTIVITY_DETAILS table function. This table function returns an XML document where the elements in the document describe the activity. In this example, the XMLTABLE function is used to return a result table from the XML output.

```sql
FROM TABLE(MON_GET_ACTIVITY_DETAILS(65592, 1, 1, -2)) AS ACTDETAILS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
'$details/db2_activity_details' PASSING XMLPARSE(DOCUMENT
ACTDETAILS.DETAILS) as "details"
COLUMNS "APP_HANDLE" BIGINT PATH 'application_handle',
"MEMBER" BIGINT PATH 'member',
"COORD_MEMBER" BIGINT PATH 'coord_member',
"LOCAL_START_TIME" VARCHAR(26) PATH 'local_start_time',
"UOW_ID" BIGINT PATH 'uow_id',
"ACTIVITY_ID" BIGINT PATH 'activity_id',
"PARENT_UOW_ID" BIGINT PATH 'parent_uow_id',
"PARENT_ACTIVITY_ID" BIGINT PATH 'parent_activity_id',
"ACTIVITY_TYPE" VARCHAR(10) PATH 'activity_type',
"NESTING_LEVEL" BIGINT PATH 'nesting_level',
"INVOCATION_ID" BIGINT PATH 'invocation_id',
"ROUTINE_ID" BIGINT PATH 'routine_id'
) AS D;
```

The query results have been divided in two parts for readability purposes.

Note: The query results have been divided in two parts for readability purposes.

The table functions mentioned previously provide a high-level description of work that is running in the system. The information that these table functions provide regarding the status of the work is limited to an activity state such as
EXECUTING. If you want to probe further to discover what exactly is occurring in a service class at a point in time, you can run the WLM_GET_SERVICE_CLASS_AGENTS table function.

In the following example, WLM_GET_SERVICE_CLASS_AGENTS is called by passing 1 for application_handle and -2 (a wildcard character) for member:

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
       SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
       SUBSTR(AGENT_STATE,1,10) AS AGENTSTATE,
       SUBSTR(REQUEST_TYPE,1,14) AS REQTYPE,
       SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
       SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS('', '', 1, -2)) AS SCDETAILS
ORDER BY APPHANDLE, MEMB, AGENT_TID
```

<table>
<thead>
<tr>
<th>APPHANDLE</th>
<th>MEMB</th>
<th>AGENT_TID</th>
<th>AGENTTYPE</th>
<th>AGENTSTATE</th>
<th>REQTYPE</th>
<th>UOW_ID</th>
<th>ACT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>COORDINATOR</td>
<td>ACTIVE</td>
<td>FETCH</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>PDBSUBAGENT</td>
<td>ACTIVE</td>
<td>SUBSECTION:1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>PDBSUBAGENT</td>
<td>ACTIVE</td>
<td>SUBSECTION:2</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

The results show a coordinator agent and a subagent on member 0 and a subagent on member 1 operating on behalf of an activity with a unit of work identifier of 1 and an activity identifier of 5. The coordinator agent information indicates that the request is a fetch request.

**Example: Obtaining point-in-time statistics from service classes**

Every activity is mapped to a service class before being executed. You can monitor the system by using the service class statistics table functions and querying all of the service classes on all of the database members to obtain point-in-time statistics.

You can use the following statement to obtain service class statistics, such as the average activity lifetime. Passing an empty string for an argument for the WLM_GET_SERVICE_SUBCLASS_STATS table function means that the result is not to be restricted by that argument. The value of the last argument, member, is -2 (a wildcard character), which means that data from all database members is to be returned.

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       CAST(COORD_ACT_LIFETIME_AVG / 1000 AS DECIMAL(9,3)) AS AVGLIFETIME,
       CAST(COORD_ACT_LIFETIME_STDDEV / 1000 AS DECIMAL(9,3)) AS STDDEVLIFETIME,
       SUBSTR(CAST(LAST_RESET AS VARCHAR(30)),1,16) AS LAST_RESET
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '', -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, MEMB
```

<table>
<thead>
<tr>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
<th>MEMB</th>
<th>AVGLIFETIME</th>
<th>STDDEVLIFETIME</th>
<th>LAST_RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>691.242</td>
<td>34.322</td>
<td>2006-07-24-11.44</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>1</td>
<td>644.740</td>
<td>22.124</td>
<td>2006-07-24-11.44</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2</td>
<td>612.431</td>
<td>43.347</td>
<td>2006-07-24-11.44</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>3</td>
<td>593.451</td>
<td>28.329</td>
<td>2006-07-24-11.44</td>
</tr>
</tbody>
</table>

**Note:** Lifetime information is only returned for those service classes that are defined with COLLECT AGGREGATE ACTIVITY DATA.
You can also use the WLM_GET_SERVICE_SUBCLASS_STATS table function to obtain the high watermark for the concurrency of coordinator activities that run in the service class on each member:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       CONCURRENT_ACT_TOP AS ACTHIGHWATERMARK
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '', -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, MEMB
```

```
SUPERCLASS_NAME SUBCLASS_NAME MEMB ACTHIGHWATERMARK
------------------- ------------------ ---- ----------------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0 10
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1 0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2 0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3 0
```

By reviewing the average lifetime and number of completed activities, you can use the output of the WLM_GET_SERVICE_SUBCLASS_STATS table function to obtain a rolled-up view of the workload on each member in the database. Significant variations in the high watermarks and averages returned by a table function might indicate a change in the workload on the system.

**Example: Aggregating data using DB2 workload management table functions**

You can perform various aggregations on table data in a DB2 workload management configuration to monitor the system and identify potential problems.

The following is an example of data aggregation that you can perform to identify problems.

**Identifying sudden increases in the number of queries running in a workload**

Assume that you have a workload called WL1. You can identify a situation in which a large number of queries are running in the workload by showing the total number of executing non-nested coordinator activities for the workload across the whole system:

```
SELECT SUBSTR(WORKLOAD_NAME,1,22) AS WLNAME,
       COUNT(*) AS TOTAL_EXE_ACT
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', -2)) AS APPS,
       TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(APPS.APPLICATION_HANDLE, -2)) AS APPACTS
WHERE WORKLOAD_NAME = 'WL1' AND
      APPS.DBPARTITIONNUM = APPS.COORD_PARTITION_NUM AND
      ACTIVITY_STATE = 'EXECUTING' AND
      NESTING_LEVEL = 0
GROUP BY WORKLOAD_NAME
```

```
WLNAME TOTAL_EXE_ACT
---------- -----------
WL1 5
```

**Example: Determining which activities are queued by a WLM threshold and their queue order**

You can use a DB2 workload manager (WLM) table function, WLM_GET_SERVICE_CLASS_AGENTS, to view which applications or activities are queued by a WLM threshold and the order of the applications or activities in the queue.
Any agent queued by a WLM threshold will have the value WLM_QUEUE in the EVENT_OBJECT column in the corresponding row returned for the agent. In addition, the AGENT_STATE_LAST_UPDATE_TIME column will contain the time that the agent entered the WLM_QUEUE state, which is the time that the agent was queued. Using this information, a simple view can be constructed to list all applications or activities that are queued by a WLM threshold, and to obtain their queue entry time.

To create a view that uses the WLM_GET_SERVICE_CLASS_AGENTS table function to obtain the WLM queue information for the examples provided here, run the following statement:

```
CREATE VIEW WLM_QUEUE_INFO (APPLICATION_HANDLE, UOW_ID, ACTIVITY_ID, THRESHOLD_NAME, QUEUE_ENTRY_TIME, MEMBER)
AS SELECT APPLICATION_HANDLE, UOW_ID, ACTIVITY_ID, VARCHAR(EVENT_OBJECT_NAME, 128), AGENT_STATE_LAST_UPDATE_TIME, MEMBER FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS(NULL,NULL,NULL,-2)) AS T WHERE EVENT_OBJECT = 'WLM_QUEUE'
```

**Note**: For TOTALSCPARTITIONCONNECTIONS thresholds, **uow_id** and **activity_id** are not applicable (NA) and, therefore, show up as NULL.

This view can be used to easily answer questions such as the following:

- How many applications or activities are currently queued by a WLM threshold?
- What is the order of the applications or activities in the WLM threshold queue?

**Example 1**

To count the number of applications queued by each queuing threshold, run the following statement:

```
SELECT VARCHAR(THRESHOLD_NAME, 30) AS THRESHOLD, COUNT(*) AS QUEUED_ENTRIES FROM WLM_QUEUE_INFO GROUP BY THRESHOLD_NAME
```

The following is a sample of the output obtained after running the preceding statement:

<table>
<thead>
<tr>
<th>THRESHOLD</th>
<th>QUEUED_ENTRIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>TH1</td>
<td>3</td>
</tr>
</tbody>
</table>

1 record(s) selected.

**Example 2**

To list all the activities that are queued in the TH1 CONCURRENTDBCoordActivities threshold and show the order of the activities in the queue determined by their queue entry time, run the following statement:

```
SELECT QUEUE_ENTRY_TIME, APPLICATION_HANDLE, UOW_ID, ACTIVITY_ID FROM WLM_QUEUE_INFO WHERE THRESHOLD_NAME = 'TH1' ORDER BY QUEUE_ENTRY_TIME ASC
```

The following is a sample of the output obtained after running the preceding statement:

<table>
<thead>
<tr>
<th>QUEUE_ENTRY_TIME</th>
<th>APPLICATION_HANDLE</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-11-09-18.08.32.583286</td>
<td>145</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Historical monitoring with WLM event monitors

DB2 workload management uses event monitors to capture information that might be of use in the future or for historical analysis.

Three event monitors are available for you to use. Each event monitor serves a different purpose:

Activity event monitor

This monitor captures information about individual activities in a service class, workload, or work class or activities that violated a threshold. The amount of data that is captured for each activity is configurable and should be considered when you determine the amount of disk space and the length of time required to keep the monitor data. A common use for activity data is to use it as input to tools such as db2advis or to use access plans (from the explain utility) to help determine table, column, and index usage for a set of queries.

You can collect information about an activity by specifying COLLECT ACTIVITY DATA for the service class, workload, or work action to which such an activity belongs or a threshold that might be violated by such an activity. The information is collected when the activity completes, regardless of whether the activity completes successfully.

Note that if an activities event monitor is active when the database deactivates, any backlogged activity records in the queue are discarded. To ensure that you obtain all activities event monitor records and that none are discarded, explicitly deactivate the activities event monitor first before deactivating the database. When an activities event monitor is explicitly deactivated, all backlogged activity records in the queue are processed before the event monitor deactivates.

Threshold violations event monitor

This monitor captures information when a threshold is violated. It indicates what threshold was violated, the activity that caused the violation, and what action was taken when it occurred.

If you specify COLLECT ACTIVITY DATA for the threshold and an activities event monitor is created and active, information is also collected about activities that violate the threshold, but this information is collected when the activity ends (either successfully or unsuccessfully).

You can obtain details about a threshold by querying the SYSCAT.THRESHOLDS view.

Statistics event monitor

This monitor serves as a low-overhead alternative to capturing detailed activity information by collecting aggregate data (for example, the number of activities completed and average execution time). Aggregate data includes histograms for a number of activity measurements including lifetime, queue time, execution time and estimated cost. You can use histograms to understand the distribution of values, identify outliers, and compute additional statistics such as averages and standard deviations. For example, histograms can help you understand the variation in lifetime that users experience. The average life time alone does not reflect what a user
experiences if there is a high degree of variability. See “Collecting workload management statistics using a statistics event monitor” on page 267 for a description of how to send statistics to the event monitor.

The following figure shows the different monitoring options available to access workload information: table functions to access real-time statistics, and activity details and historical information captured as efficient aggregates or as details about individual activities:
Unlike statement, connection, and transaction event monitors, the activity, statistics, and threshold violations event monitors do not have event conditions (that is, conditions specified on the WHERE keyword of the CREATE EVENT MONITOR statement). Instead, these event monitors rely on the attributes of service classes, workloads, work classes, and thresholds to determine whether these objects send their activity information or aggregate information to these monitors.

Figure 46. Workload management with monitoring
Typically, event monitors write data to either tables or files. You need to prune these tables or files periodically because they are not automatically pruned.

You can use the `wlmevmon.ddl` script in the `sqllib/misc` directory to create and enable three event monitors called `DB2ACTIVITIES`, `DB2STATISTICS`, and `DB2THRESHOLDVIOLATIONS`. If necessary, modify the script to change the table space or other parameters.

Example

**Example: Identify queries with a large estimated cost using the statistics event monitor:** You suspect that your database workload occasionally includes large, expensive queries, possibly due to the poor optimization of the queries themselves. You want to identify these queries so that you can prevent them from consuming excessive resources on your system, with a long-term goal of perhaps rewriting some of the queries to improve performance. The statistics event monitor provides you with a low-overhead way to measure the estimated cost of your queries which you can then use to determine what the maximum acceptable estimated cost for a query on your data server should be. A query that is poorly optimized is typically distinguished by a large estimated cost that is many times larger than the estimated cost of most other queries.

To get started, you need to create and activate a statistics event monitor and to start collecting extended aggregate activity data for the service class where the queries run:

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

In this example, all queries run in the SYSDEFAULTSUBCLASS subclass of the SYSDEFAULTUSERCLASS service class, which you can alter to collect the required data:

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

A full day of work might provide a reasonable approximation of the range of queries your data server typically processes. At the end of the day, you can copy the statistics collected from memory to the statistics event monitor by running the WLM_COLLECT_STATS stored procedure:

```sql
CALL WLM_COLLECT_STATS()
```

Included with the different statistics written to the event monitor tables are the estimated cost statistics of queries. To see them, you can query the service class statistics table `SCSTATS_DB2STATISTICS`:

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

<table>
<thead>
<tr>
<th>STATISTICS_TIMESTAMP</th>
<th>COORD_ACT_EST_COST_AVG</th>
<th>COST_ESTIMATE_TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-09-03-09.49.04.455979</td>
<td>169440</td>
<td>13246445</td>
</tr>
</tbody>
</table>

1 record(s) selected.
The output shows that the average query has an estimated cost in the range of hundreds of thousands of timerons, and that the largest queries have estimated costs larger than ten million timerons. You can confirm that queries of ten million or more timerons are outliers by looking at the estimated cost histogram, which was generated at the same time that the average and high watermarks shown in the output were written to the event monitor table. You can look at the histogram by querying the HISTOGRAMBIN_DB2STATISTICS table as follows:

```sql
SELECT STATISTICS_TIMESTAMP, TOP, NUMBER_IN_BIN
FROM HISTOGRAMBIN_DB2STATISTICS HIST,
     SYSCAT.SERVICECLASSES SC
WHERE HIST.SERVICE_CLASS_ID = SC.SERVICECLASSID
  AND SC.PARENTSERVICECLASSNAME = 'SYSDEFAULTUSERCLASS'
  AND SC.SERVICECLASSNAME = 'SYSDEFAULTSUBCLASS'
  AND HISTOGRAM_TYPE = 'COORDACTESTCOST'
```

<table>
<thead>
<tr>
<th>STATISTICS_TIMESTAMP</th>
<th>TOP</th>
<th>NUMBER_IN_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-09-03-09.49.04</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>68</td>
<td>5</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>103</td>
<td>22</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>158</td>
<td>14</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>241</td>
<td>54</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>369</td>
<td>2</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>562</td>
<td>142</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>858</td>
<td>21</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>1309</td>
<td>123</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>1997</td>
<td>512</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>3046</td>
<td>643</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>4647</td>
<td>201</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>7089</td>
<td>875</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>10813</td>
<td>1445</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>16493</td>
<td>5386</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>25157</td>
<td>2409</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>38373</td>
<td>8940</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>58532</td>
<td>9820</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>89280</td>
<td>2149</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>136181</td>
<td>798</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>207720</td>
<td>2411</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>316840</td>
<td>14909</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>483283</td>
<td>9831</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>737162</td>
<td>1451</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>112449</td>
<td>213</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>1715085</td>
<td>24</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>2616055</td>
<td>1</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>3990325</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>6086529</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>9283913</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>14160950</td>
<td>3</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>21600000</td>
<td>0</td>
</tr>
<tr>
<td>2008-09-03-09.49.04</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

In the histogram, the value in the number_in_bin column for queries whose top is greater than 2616055 is zero until top reaches 14160950, where the number_in_bin becomes 3. These three queries are outliers and can be controlled with an ESTIMATEDSQLCOST threshold set to trigger if the estimated cost of a query
exceeds 10 million timerons which you can use to prevent such activities from executing and to monitor them more closely.

**Example: Using the threshold violations event monitor:** To control activities of a certain estimated cost, you want to define an ESTIMATEDSQLCOST threshold on your workload that applies only to that subset of your total workload exceeding a certain estimated cost. Having looked at the estimated cost histogram, you determined that activities with an estimated cost in the range of 0 to under 3 million timerons occur frequently and that activities with an estimated cost over 10 million timerons occur rarely (perhaps only a few times a day and perhaps always due to some flaw in the query, such as the use of a Cartesian join).

To verify that a threshold of 10 million timerons is effective in stopping those few activities a day that should not be allowed to run, you can create and activate a threshold event monitor:

```
CREATE THRESHOLD TH1
   FOR DATABASE ACTIVITIES
   ENFORCEMENT DATABASE
   WHEN ESTIMATEDSQLCOST > 10000000
   STOP EXECUTION

CREATE EVENT MONITOR DB2THRESHOLDVIOLATIONS
   FOR THRESHOLD VIOLATIONS
   WRITE TO TABLE

SET EVENT MONITOR DB2THRESHOLDVIOLATIONS STATE 1
```

After the end of the day, you can see what threshold violations occurred by querying the threshold violations table:

```
SELECT THRESHOLDID,
       SUBSTR(THRESHOLD_PREDICATE, 1, 20) PREDICATE,
       TIME_OF_VIOLATION,
       THRESHOLD_MAXVALUE,
       THRESHOLD_ACTION
FROM THRESHOLDVIOLATIONS_DB2THRESHOLDVIOLATIONS
ORDER BY TIME_OF_VIOLATION, THRESHOLDID
```

<table>
<thead>
<tr>
<th>THRESHOLDID</th>
<th>PREDICATE</th>
<th>TIME_OF_VIOLATION</th>
<th>THRESHOLD_MAXVALUE</th>
<th>THRESHOLD_ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EstimatedSQLCost</td>
<td>2008-09-02-22.39.10.000000</td>
<td>10000000</td>
<td>Stop</td>
</tr>
</tbody>
</table>

1 record(s) selected.

**Example: Using the activity event monitor**

The previous example showed how you can collect threshold information in an event monitor table to confirm that activities with a large estimated cost are being prevented from executing by a threshold. After seeing these threshold violations, you want to determine what the SQL statement texts producing these large queries are, so that you can use the explain facility to determine if an index is needed on the tables being queried.

Collecting this additional information requires creating and activating an activity event monitor and altering the threshold to turn on activity collection with details:

```
CREATE EVENT MONITOR DB2ACTIVITIES
   FOR ACTIVITIES WRITE TO TABLE

SET EVENT MONITOR DB2ACTIVITIES STATE 1
```
ALTER THRESHOLD TH1
WHEN EXCEEDED
COLLECT ACTIVITY DATA WITH DETAILS

When you query the threshold violations table again after another business day has passed, you can perform a join with the ACTIVITYSTMT_DB2ACTIVITIES table to see the SQL statement text of any activity that violated the threshold:

```
SELECT THRESHOLDID,
     SUBSTR(THRESHOLD_PREDICATE, 1, 20) PREDICATE,
     TIME_OF_VIOLATION,
     SUBSTR(STMT_TEXT,1,70) STMT_TEXT
FROM THRESHOLDVIOLATIONS_DB2THRESHOLDVIOLATIONS TV,
     ACTIVITYSTMT_DB2ACTIVITIES A
WHERE TV.APPL_ID = A.APPL_ID
     AND TV.UOW_ID = A.UOW_ID
     AND TV.ACTIVITY_ID = A.ACTIVITY_ID
```

<table>
<thead>
<tr>
<th>THRESHOLDID</th>
<th>PREDICATE</th>
<th>TIME_OF_VIOLATION</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EstimatedSQLCost</td>
<td>2008-09-02-23.04.49.000000</td>
<td>select count(*) from syscat.tables,syscat.tables,syscat.tables</td>
</tr>
</tbody>
</table>

Available monitoring data

Monitoring data is available from workloads, service subclasses and service superclasses, work classes, and threshold queues. You can use this data to diagnose and correct problems and for performance tuning.

Workload monitoring data

The following figure shows the monitoring information that is available for workloads. You can collect workload statistics and information about activities that run in the workloads using event monitors. For workloads, you can also obtain aggregate activity statistics. You can access workload statistics and information about workload occurrences in real time using table functions.

![Figure 47. Monitoring data that is available for workloads](image-url)

Service class monitoring data

The following figure shows the monitoring information that is available for service classes. You can collect statistics for service subclasses and service superclasses. For
service subclasses, you can also obtain aggregate activity and request statistics, and information about activities that run in the service subclass. You can access service superclass and service subclass statistics and information about agents running in a particular service class in real time using table functions.

**Work class monitoring data**

The following figure shows the monitoring information that is available for work classes. You can collect work class statistics and information about activities that are associated with a particular work class. You can access work class statistics in real time using table functions.

**Threshold monitoring data**

The following figure shows the monitoring information that is available for thresholds. You can obtain information about threshold violations, the activities that caused the threshold violations, and queuing statistics (for queuing thresholds). You can access queuing threshold statistics in real time using table functions.
functions.

The following stored procedures are available for use with DB2 workload management:

**WLM_CANCEL_ACTIVITY(application_handle, uow_id, activity_id)**

Use this stored procedure to cancel a running or queued activity. You identify the activity by its application handle, unit of work identifier, and activity identifier. You can cancel any type of activity. The application with the cancelled activity receives the error SQL4725N.

**WLM_CAPTURE_ACTIVITY_IN_PROGRESS(application_handle, uow_id, activity_id)**

Use this stored procedure to send information about an individual activity that is currently executing to the activities event monitor. This stored procedure sends the information immediately, rather than waiting until the activity completes.

**WLM_COLLECT_STATS()**

Use this stored procedure to collect and reset statistics for DB2 workload management objects. All statistics tracked for service classes, workloads, threshold queues, and work action sets are sent to the active statistics event monitor (if one exists) and reset. If there is no active statistics event monitor, the statistics are only reset, but not collected.

**WLM_SET_CLIENT_INFO(client_userid, client_wrkstname, client_applname, client_acctstr, client_workload)**

Use this procedure to set the client information attributes used at the data server to record the identity of the application or end-user currently using
the connection. In cases where middleware exists between applications or users and your data server, use the WLM_SET_CLIENT_INFO procedure to set distinguishing connection attributes explicitly.

---

**Statistics for DB2 workload management objects**

Statistics are maintained for DB2 workload management objects including service classes, work classes, workloads, and threshold queues. These statistics reside in memory and can be viewed in real-time using DB2 workload management statistics table functions, or the statistics can be collected and sent to a statistics event monitor where they can be viewed later for historical analysis.

Note that you can also obtain monitoring metrics through the statistics event monitor. These are not discussed in this topic, which covers only those statistics that are specific to DB2 workload management.

When statistics are sent to the event monitor, the values in memory are reset to prevent duplicate data from being collected on subsequent collection intervals. Because the DB2 workload management statistics table functions report the current in-memory values, following a collection they report the reset values. The DB2 workload management table functions report only a subset of the statistics. To view the full set of statistics, you must collect the statistics and send them to a statistics event monitor.

**Aggregate activity data statistics collection**

The following statistics are maintained on the given objects on each member, regardless of the value of the COLLECT AGGREGATE ACTIVITY DATA option specified for those objects when they are created or altered.
Table 58. Statistics collected for database objects regardless of COLLECT AGGREGATE ACTIVITY DATA setting

<table>
<thead>
<tr>
<th>Database object</th>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service subclasses</td>
<td>1. Concurrent activity top</td>
<td>1. Use this activity concurrency high watermark to determine the highest concurrency of activities (including nested activities) reached on a member for a service class in the time interval for which the statistic is collected.</td>
</tr>
<tr>
<td></td>
<td>(concurrent_act_top)</td>
<td>2. Use this statistic to determine how much work is being performed in a service class.</td>
</tr>
<tr>
<td></td>
<td>2. Coordinator activities</td>
<td>3. Use this statistic, which measures the unsuccessful completion of activities, to determine how healthy the system is. Activities can be aborted because of cancellation, errors, or reactive thresholds.</td>
</tr>
<tr>
<td></td>
<td>completed total</td>
<td>4. Use this rejected non-nested coordinator activity count, which measures the rejection of activities, to obtain an indication of the usefulness of the rejection policy. Activities are counted as rejected when they violate a predictive threshold that has an action of STOP EXECUTION or when they are prevented from executing by a work action.</td>
</tr>
<tr>
<td></td>
<td>(coord_act_completed_total)</td>
<td>5. Use this statistic to determine the number of requests that are currently executing in a service class.</td>
</tr>
<tr>
<td></td>
<td>3. Coordinator activities</td>
<td>6. Use these statistics to determine the number of activities that are remapped into or out of a service subclass as part of the priority aging of ongoing activities.</td>
</tr>
<tr>
<td></td>
<td>aborted total</td>
<td>(coord_act_aborted_total)</td>
</tr>
<tr>
<td></td>
<td>4. Coordinator activities</td>
<td>(coord_act_rejected_total)</td>
</tr>
<tr>
<td></td>
<td>rejected total</td>
<td>(coord_act_rejected_total)</td>
</tr>
<tr>
<td></td>
<td>5. The number of active requests</td>
<td>(num_requests_active)</td>
</tr>
<tr>
<td></td>
<td>6. The number of activities</td>
<td>(act_remapped_in and act_remapped_out)</td>
</tr>
<tr>
<td></td>
<td>mapped in and the number of</td>
<td>activities mapped out (act_remapped_in and act_remapped_out)</td>
</tr>
<tr>
<td></td>
<td>activities mapped out</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service superclasses</td>
<td>Concurrent connection top</td>
<td>1. Use this coordinator connection concurrency high watermark to tune a connection concurrency threshold.</td>
</tr>
</tbody>
</table>
Table 58. Statistics collected for database objects regardless of COLLECT AGGREGATE ACTIVITY DATA setting (continued)

<table>
<thead>
<tr>
<th>Database object</th>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workloads</td>
<td>1. Concurrent workload occurrences top (concurrent_wlo_top)</td>
<td>1. Use this workload occurrence high watermark to identify the maximum number of concurrent workload occurrences and to help set or tune a workload occurrence concurrency threshold if the number of concurrently executing workload occurrences is too high (that is, too many applications that are associated with the same workload definition are running on the system at the same time).</td>
</tr>
<tr>
<td></td>
<td>2. Concurrent workload occurrences activity top (concurrent_wlo_act_top)</td>
<td>2. Use this element to know the highest number of concurrent activities reached on a member for any occurrence of this workload in the time interval collected.</td>
</tr>
<tr>
<td></td>
<td>3. Coordinator activities completed total (coord_act_completed_total)</td>
<td>3. Use this statistic, which measures the rate of successful completion of activities, to obtain an indication of the health of the system.</td>
</tr>
<tr>
<td></td>
<td>4. Coordinator activities aborted total (coord_act_aborted_total)</td>
<td>4. Use this statistic, which measures the unsuccessful completion of activities, to determine how healthy the system is. Activities can be aborted due to cancellation, errors, or reactive thresholds.</td>
</tr>
<tr>
<td></td>
<td>5. Coordinator activities rejected total (coord_act_rejected_total)</td>
<td>5. Use this statistic, which measures the rate of rejection of activities, to determine the usefulness of a rejection policy. Activities are counted as rejected when they violate a predictive threshold that has an action of STOP EXECUTION or when they are prevented from executing by a work action.</td>
</tr>
<tr>
<td></td>
<td>6. Workload occurrences completed total (wlo_completed_total)</td>
<td>6. Use this statistic to determine how many occurrences of a workload complete in a specific period of time.</td>
</tr>
<tr>
<td></td>
<td>7. Activities total (act_total)</td>
<td>7. Use this statistic to determine the effectiveness of the work action set and determine the relative percentages of the types of activities on the system.</td>
</tr>
</tbody>
</table>
Table 58. Statistics collected for database objects regardless of COLLECT AGGREGATE ACTIVITY DATA setting (continued)

<table>
<thead>
<tr>
<th>Database object</th>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work class (through a work action)</td>
<td>Queue assignments total</td>
<td>Use this statistic to determine whether excessive queuing is occurring, or whether the right number of activities are being queued (that is, whether the concurrency threshold is too restrictive or not restrictive enough).</td>
</tr>
<tr>
<td></td>
<td>(queue_assignments_total)</td>
<td></td>
</tr>
<tr>
<td>Threshold queues</td>
<td>1. Queue size top (queue_size_top)</td>
<td>1. Use this statistic to help determine the maximum queue size and to identify whether the queue size is sufficient.</td>
</tr>
<tr>
<td></td>
<td>2. Queue time total (queue_time_total)</td>
<td>2. Use this statistic to determine how much time activities are spending in the queue and whether that time is excessive.</td>
</tr>
</tbody>
</table>

When you set the value of the COLLECT AGGREGATE ACTIVITY DATA option to BASE for a service subclass, workload, or a work class (through a work action), some of the following statistics are also collected, or the corresponding histograms are generated for each member. Use the averages to quickly understand where activities are spending most of their time (for example, queued or executing) and the response time (lifetime). You can also use the averages to tune the histogram templates. That is you can compare a true average with the average computed from a histogram, and if the average from the histogram deviates from the true average, consider altering the histogram template for the corresponding histogram, using a set of bin values that are more appropriate for your data.

Table 59. Statistics or histograms collected when COLLECT AGGREGATE ACTIVITY DATA is set to BASE

<table>
<thead>
<tr>
<th>Statistic or histogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average request execution time (request_exec_time_avg)</td>
<td>Use this statistic to determine the arithmetic mean of the execution times for requests associated with a service class.</td>
</tr>
<tr>
<td>Average coordinator activity lifetime (coord_act_lifetime_avg)</td>
<td>Use this statistic to determine the arithmetic mean of the lifetime for non-nested coordinator activities associated with a service class, workload or a work class.</td>
</tr>
<tr>
<td>Average coordinator activity execution time (coord_act_exec_time_avg)</td>
<td>Use this statistic to determine the arithmetic mean of execution time for non-nested coordinator activities associated with a service class, workload or a work class.</td>
</tr>
<tr>
<td>Average coordinator activity queue time (coord_act_queue_time_avg)</td>
<td>Use this statistic to determine the arithmetic mean of the queue time for non-nested coordinator activities associated with a service class, workload or a work class.</td>
</tr>
<tr>
<td>Cost estimate top (cost_estimate_top)</td>
<td>Use this statistic to tune estimated cost thresholds.</td>
</tr>
<tr>
<td>Actual rows returned top (rows_returned_top)</td>
<td>Use the information to tune the actual rows returned thresholds.</td>
</tr>
</tbody>
</table>
Table 59. Statistics or histograms collected when COLLECT AGGREGATE ACTIVITY DATA is set to BASE (continued)

<table>
<thead>
<tr>
<th>Statistic or histogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate temporary table space top (agg_temp_tablespace_top)</td>
<td>Use this statistic to tune aggregate system temporary table space usage. This statistic is monitored only if you define a threshold for aggregate temporary table space usage. For any given service subclass, this statistic is monitored whenever you define a AGGSQLTEMPSPACE threshold on the service subclass itself, or when you define a similar threshold on any service subclass within the same superclass.</td>
</tr>
<tr>
<td>Temporary table space top (temp_tablespace_top)</td>
<td>Use this statistic to tune temporary table space usage thresholds. This statistic is monitored only if you define a threshold for temporary table space usage. For any given service subclass, this threshold is also monitored whenever you define a AGGSQLTEMPSPACE threshold on the service subclass itself, or when you define a similar threshold on any service subclass within the same superclass.</td>
</tr>
<tr>
<td>Coordinator activity lifetime (CoordActLifetime) histogram</td>
<td>Use this histogram to obtain a view of overall system performance. This histogram collects the time duration (in milliseconds) between the activity arrival and end time for non-nested coordinator activities. If the activity is a routine that leaves a cursor open after it ends, the lifetime histogram does not count the lifetime of the cursor toward the lifetime of the routine that is the parent of the cursor.</td>
</tr>
</tbody>
</table>
Table 59. Statistics or histograms collected when COLLECT AGGREGATE ACTIVITY DATA is set to BASE (continued)

<table>
<thead>
<tr>
<th>Statistic or histogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinator activity execution time (CoordActExecTime) histogram</td>
<td>Use this histogram to measure the impact of changes to the system that affect execution time. This histogram collects the execution time (in milliseconds) for non-nested coordinator activities. The execution time is calculated as follows: • For cursors, the execution time is the combined time for the open cursor request, any fetches, and the close cursor request. Time when the cursor is idle is not counted towards the execution time. • For routines, the execution time is from the start to the end of the routine invocation. If any cursors are left open by the routine after it ends, the lifetimes of these cursors are not counted towards the routine execution time. • For all other activities, the execution time is the difference between the activity lifetime and the time that the activity spends queued.</td>
</tr>
<tr>
<td>Coordinator activity queue time (CoordActQueueTime) histogram</td>
<td>Use this histogram to measure the impact of queuing thresholds on activities. This histogram collects the amount of time (in milliseconds) that non-nested coordinator activities spend queued.</td>
</tr>
</tbody>
</table>

When you set the value of the COLLECT AGGREGATE ACTIVITY DATA option to EXTENDED for a service subclass, workload or a work class, the following system statistics are collected or histograms are generated for each member for the corresponding service class or work class (through a work action). Use the averages to quickly understand the average rate of arrival of activities (arrival rate is the inverse of inter-arrival time) and the expense of activities (estimated cost). You can also use the averages to tune the histogram templates. That is you can compare a true average with the average computed from a histogram, and if the average from the histogram deviates from the true average, consider altering the histogram template for the corresponding histogram, using a set of bin values that are more appropriate for your data. EXTENDED statistics are useful for more detailed performance modelling. Also see “Workload management performance modelling” on page 281.
Table 60. Statistics or histograms collected when COLLECT AGGREGATE ACTIVITY DATA is set to EXTENDED

<table>
<thead>
<tr>
<th>Statistic or histogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinator activity estimated cost average (coord_act_est_cost_avg)</td>
<td>Use this statistic to determine the arithmetic mean of the estimated costs of coordinator DML activities at nesting level 0 that are associated with this service subclass, workload or work class since the last statistics reset.</td>
</tr>
<tr>
<td>Non-nested coordinator activity inter-arrival time average (coord_act_interarrival_time_avg)</td>
<td>Use this statistic to determine the arithmetic mean of the time between the arrival of one coordinator activity at nesting level 0 that is associated with this service class, workload or work class and the next coordinator activity to arrive. The average is computed since the last statistics reset.</td>
</tr>
<tr>
<td>Coordinator activity estimated cost (CoordActEstCost) histogram.</td>
<td>Use this histogram to obtain an approximate service time distribution. This histogram collects the estimated cost (in timerons) for non-nested coordinator activities. This data is useful for modelling your system or for inputting into performance-modelling applications.</td>
</tr>
<tr>
<td>Coordinator activity inter-arrival time (CoordActInterArrivalTime) histogram.</td>
<td>Use this histogram to obtain the inter-arrival time distribution for non-nested coordinator activities. This histogram collects the inter-arrival time (in milliseconds) for non-nested coordinator activities. This data is useful for modelling your system or for inputting into performance-modeling applications.</td>
</tr>
</tbody>
</table>

The following table provides a reference for which activity statistics are collected for each DB2 workload management object and includes all aggregate statistics available to you from both table functions and event monitors. Some statistics are always collected for some objects. Other statistics are only collected when a particular COLLECT AGGREGATE option is specified. For aggregate activity statistics, if COLLECT AGGREGATE ACTIVITY DATA EXTENDED is specified, all the BASE aggregate activity statistics are also collected.
### Table 61. Aggregate activity statistics collection for DB2 workload management objects

<table>
<thead>
<tr>
<th>Object type</th>
<th>Activity statistics always collected by default</th>
<th>Activity statistics collected when you specify COLLECT AGGREGATE ACTIVITY DATA BASE</th>
<th>Activity statistics collected when you specify COLLECT AGGREGATE ACTIVITY DATA EXTENDED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service subclass</td>
<td>act_remapped_in, act_remapped_out, concurrent_act_top, coord_act_completed_total, coord_act_rejected_total, coord_act_aborted_total</td>
<td>coord_act_exec_time_avg, coord_act_lifetime_avg, coord_act_queue_time_avg, coord_act_lifetime_stdddev, coord_act_exec_time_stdddev, coord_act_queue_time_stdddev, CoordActLifetime histogram, CoordActExecTime histogram, CoordActQueueTime histogram</td>
<td>coord_act_est_cost_avg, coord_act_interarrival_time_avg, CoordActEstCost histogram, CoordActInterArrivalTime histogram</td>
</tr>
<tr>
<td>Service superclass</td>
<td>concurrent_connection_top</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Workload</td>
<td>concurrent_wlo_act_top, concurrent_wlo_top, coord_act_aborted_total, coord_act_completed_total, coord_act_rejected_total, wlo_completed_total</td>
<td>coord_act_exec_time_avg, coord_act_lifetime_avg, coord_act_queue_time avg, coord_act_lifetime_stdddev, coord_act_exec_time_stdddev, coord_act_queue_time_stdddev, CoordActLifetime histogram, CoordActExecTime histogram, CoordActQueueTime histogram</td>
<td>coord_act_est_cost_avg, coord_act_interarrival_time_avg, CoordActEstCost histogram, CoordActInterArrivalTime histogram</td>
</tr>
<tr>
<td>Work class (through a work action)</td>
<td>act_total</td>
<td>agg_temp_tablespace_top, coord_act_lifetime_top, coord_act_lifetime_avg, coord_act_queue_time_avg, coord_act_lifetime_stdddev, coord_act_exec_time_stdddev, CoordActLifetime histogram, CoordActExecTime histogram, CoordActQueueTime histogram</td>
<td>coord_act_est_cost_avg, coord_act_interarrival_time_avg, CoordActEstCost histogram, CoordActInterArrivalTime histogram</td>
</tr>
<tr>
<td>Threshold</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Threshold queue</td>
<td>queue_assignments_total, queue_size_top, queue_time_total</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Aggregate request data statistics collection

When you set the value of the COLLECT AGGREGATE REQUEST DATA option for a service subclass to BASE, the following statistics are maintained for the service subclass.

*Table 62. Statistics or histograms collected when COLLECT AGGREGATE REQUEST DATA is set to BASE*

<table>
<thead>
<tr>
<th>Statistic or histogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request execution time average (request_exec_time_avg)</td>
<td>Use this statistic to quickly understand the average amount of time that is spent processing each request on a member and to help tune the histogram template for the corresponding request execution time histogram.</td>
</tr>
</tbody>
</table>
Table 62. Statistics or histograms collected when COLLECT AGGREGATE REQUEST DATA is set to BASE (continued)

<table>
<thead>
<tr>
<th>Statistic or histogram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request execution time (ReqExecTime) histogram</td>
<td>Use this histogram to understand where work is being performed and whether the distribution of work across members is uniform.</td>
</tr>
<tr>
<td></td>
<td>This histogram indicates the volume of work executing in a service class and the distribution of this work across database members. The execution time (in milliseconds) for requests is collected in a histogram for each member and for all requests.</td>
</tr>
<tr>
<td></td>
<td>This histogram includes requests on the coordinator member, and any subrequests on both coordinator and non-coordinator members (like RPC requests or SMP subagent requests). Requests included may or may not be associated with an activity: Both PREPARE and OPEN requests are included in this histogram, for example, but while OPEN requests are always associated with a cursor activity, PREPARE requests are not part of any activity.</td>
</tr>
<tr>
<td></td>
<td>The request execution time approximates the effort spent by agents working in a service class. For example, coordinator activity counts might show that most user activities originate on one member, but as part of processing the activities, the coordinator agent might be sending subrequests to another member that performs most of the work.</td>
</tr>
<tr>
<td></td>
<td>The request execution time histogram can be useful in determining the size of requests sent to a member, that is, whether the work that is sent to the member consists of mostly small requests or mostly large requests or whether there is no specific distribution.</td>
</tr>
<tr>
<td></td>
<td>Request execution time histograms should not be used for activity response time analysis, because activities may be composed of a number of requests and subrequests, because there is no one-to-one mapping between request and activity execution time, and because not all requests are associated with activities.</td>
</tr>
</tbody>
</table>

The following table provides a reference for which request statistics are collected for each DB2 workload management object and includes all aggregate statistics available to you from both table functions and event monitors. Some statistics are always collected for some objects. Other statistics are only collected when the COLLECT AGGREGATE REQUEST DATA option is specified.
Statistics collection and monitoring with remapped activities

How you collect statistics and how you monitor are both affected by the dynamic remapping of activities between service subclasses.

Remapping occurs when an activity that is executing in one service class is moved to another service class to continue execution. This remapping can be done using thresholds, such as CPUTIMEINSC, and can be an integral part of your workload management configuration, as is the case with the priority aging approach.

Statistics affected by remapped activities

As one exception to the rule, the activity interarrival time, estimated cost, and queue time are all associated with the subclass in which an activity starts running, rather than with the subclass in which the activity finishes running. Because a remapped activity affects the statistics collection of both subclasses, a different number of activities can be counted in an interarrival time, an estimated cost, or a queue-time histogram than in a lifetime or execution-time histogram.

For example, consider an activity that starts running in service subclass A and later is remapped to service subclass B, in which it finishes running. The estimated cost of this activity is associated with service subclass A, but its lifetime is associated with service subclass B. As a result, for subclass A, the estimated cost histogram has one more element counted in it than the lifetime histogram has counted in it, and for service subclass B, the lifetime histogram has one more element counted in it than the estimated cost histogram has counted in it.

As a second exception to the rule, the monitor element concurrent_act_top can be updated in and attributed to any subclass that an activity passes through. In addition to being incremented when an activity begins and decremented when an activity ends, the monitor element is incremented when an activity is mapped to the subclass and is decremented when an activity is mapped out of the subclass (mapped to a different subclass).

Statistics about activity remapping

You can use two monitor elements to count the number of activities entering or leaving a service subclass because of a remapping action: act_remapped_in and act_remapped_out. The act_remapped_in and act_remapped_out monitor elements
count the number of activities for any given subclass at any partition that were mapped into or out of the subclass since the last reset. You can use these monitor elements to validate that the remapping of activities between service subclasses is occurring as expected.

To determine the source and destination service subclasses targeted by a remapping action, you can refer to the threshold violation event monitor record, which includes a destination service class ID (destination_service_class_id). You can also determine the source service class by using the threshold violation record.

**Monitoring with activity remapping**

Remapping activities to different subclasses affects how you monitor these activities. To ensure that all statistics are collected for an activity that starts in one service class and finishes in another because of remapping, turn on aggregate activity data collection for both the service subclass in which the activity starts running and the service subclass in which the activity finishes running when you create or alter the service classes. If you turn on aggregate activity data collection for only the service subclass in which the activity started, the activity contributes only to queue time statistics and, in the case of extended statistics, to the estimated cost and interarrival time statistics. If you turn on aggregate activity data collection for only the service subclass in which the activity finishes running, the activity contributes only to lifetime and execution time statistics, regardless of whether the option is COLLECT AGGREGATE DATA BASE or COLLECT AGGREGATE DATA EXTENDED when you issue the CREATE SERVICE CLASS or ALTER SERVICE CLASS statement.

The following tables summarize how statistics collection is affected by remapping and collection settings.

*Table 64. Effect of the COLLECT AGGREGATE DATA BASE option on aggregate statistics collection for subclasses involved in remapping*

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Starting subclass collection setting and ending subclass collection setting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NONE and NONE</td>
</tr>
<tr>
<td>Lifetime</td>
<td>Not collected</td>
</tr>
<tr>
<td>Queue time</td>
<td>Not collected</td>
</tr>
<tr>
<td>Execution time</td>
<td>Not collected</td>
</tr>
</tbody>
</table>

*Table 65. Effect of the COLLECT AGGREGATE DATA EXTENDED option on aggregate statistics collection for subclasses involved in remapping*

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Starting subclass collection setting and ending subclass collection setting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NONE and NONE</td>
</tr>
<tr>
<td>Lifetime</td>
<td>Not collected</td>
</tr>
<tr>
<td>Queue time</td>
<td>Not collected</td>
</tr>
<tr>
<td>Execution time</td>
<td>Not collected</td>
</tr>
<tr>
<td>Inter-arrival time</td>
<td>Not collected</td>
</tr>
<tr>
<td>Estimated cost</td>
<td>Not collected</td>
</tr>
</tbody>
</table>
Table 66. Effect of mixing the COLLECT AGGREGATE DATA BASE and the COLLECT AGGREGATE DATA EXTENDED options on aggregate statistics collection for subclasses involved in remapping

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Starting subclass collection setting and ending subclass collection setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifetime</td>
<td>BASE and EXTENDED</td>
</tr>
<tr>
<td>Queue time</td>
<td>EXTENDED and BASE</td>
</tr>
<tr>
<td>Execution time</td>
<td>Collected</td>
</tr>
<tr>
<td>Inter-arrival time</td>
<td>Collected</td>
</tr>
<tr>
<td>Estimated cost</td>
<td>Not collected</td>
</tr>
</tbody>
</table>

**Histograms in workload management**

A histogram is a collection of bins, which are containers for collecting discrete ranges of data. Histograms are useful for a variety of workload analysis and performance-tuning tasks.

DB2 workload management histograms have a fixed number of 41 bins. The 40th bin contains the highest defined value for the histogram, and the 41st bin is for values that are beyond the highest defined value. Each bin represents a specific range of values, and the bin ranges follow a logarithmic scale, with each bin representing a progressively larger range as one progresses from the 1st to 40th bin. The following figure shows a histogram of activity lifetimes that are plotted using a bar chart.

The activity lifetime histogram corresponds to the following data. Each count represents the number of activities whose lifetimes (in milliseconds) are within the range of the low bin value to the high bin value. For example, 156 activities had a lifetime in the range of 68 milliseconds to 103 milliseconds.
You can use histograms for a number of different purposes. For example, you can use them to see the distribution of values, use them to identify outlying values, or use them to compute averages and standard deviations. See “Scenario: Tuning a DB2 workload management configuration when capacity planning information is unavailable” on page 362 and “Example: Computing averages and a standard deviation from histograms in a DB2 workload management configuration” on page 265 for examples of how to use histograms to better understand and characterize your workload.

In a multimember database environment, histograms are collected on each member. Histogram bins have the same range of values on all database members, with specific counts per bin per member. You can use the bins to analyze information on a per-member basis. You can also combine the histograms from all database members by adding the counts in the corresponding bins and use this single histogram to obtain a global view of the data, which you can then use for tasks such as calculating the global average and standard deviation.

Histograms are available for service subclasses, workloads, and work classes, through work actions. Histograms are collected for these objects when you specify one of the COLLECT AGGREGATE ACTIVITY DATA, COLLECT AGGREGATE ACTIVITY DATA command.

### Table: Histogram Bin Counts

<table>
<thead>
<tr>
<th>Low Bin</th>
<th>High Bin</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>29</td>
<td>10</td>
</tr>
<tr>
<td>29</td>
<td>44</td>
<td>15</td>
</tr>
<tr>
<td>44</td>
<td>68</td>
<td>45</td>
</tr>
<tr>
<td>68</td>
<td>103</td>
<td>156</td>
</tr>
<tr>
<td>103</td>
<td>158</td>
<td>65</td>
</tr>
<tr>
<td>158</td>
<td>241</td>
<td>23</td>
</tr>
<tr>
<td>241</td>
<td>369</td>
<td>0</td>
</tr>
<tr>
<td>369</td>
<td>562</td>
<td>0</td>
</tr>
<tr>
<td>562</td>
<td>858</td>
<td>0</td>
</tr>
<tr>
<td>858</td>
<td>1309</td>
<td>0</td>
</tr>
<tr>
<td>1309</td>
<td>1997</td>
<td>0</td>
</tr>
<tr>
<td>1997</td>
<td>3046</td>
<td>0</td>
</tr>
<tr>
<td>3046</td>
<td>4647</td>
<td>0</td>
</tr>
<tr>
<td>4647</td>
<td>7089</td>
<td>0</td>
</tr>
<tr>
<td>7089</td>
<td>10813</td>
<td>0</td>
</tr>
<tr>
<td>10813</td>
<td>16493</td>
<td>0</td>
</tr>
<tr>
<td>16493</td>
<td>25157</td>
<td>0</td>
</tr>
<tr>
<td>25157</td>
<td>38373</td>
<td>0</td>
</tr>
<tr>
<td>38373</td>
<td>58532</td>
<td>0</td>
</tr>
<tr>
<td>58532</td>
<td>89280</td>
<td>0</td>
</tr>
<tr>
<td>89280</td>
<td>136181</td>
<td>0</td>
</tr>
<tr>
<td>136181</td>
<td>207720</td>
<td>0</td>
</tr>
<tr>
<td>207720</td>
<td>316840</td>
<td>0</td>
</tr>
<tr>
<td>316840</td>
<td>483283</td>
<td>3</td>
</tr>
<tr>
<td>483283</td>
<td>737162</td>
<td>0</td>
</tr>
<tr>
<td>737162</td>
<td>1124409</td>
<td>0</td>
</tr>
<tr>
<td>1124409</td>
<td>1715085</td>
<td>0</td>
</tr>
<tr>
<td>1715085</td>
<td>2616055</td>
<td>0</td>
</tr>
<tr>
<td>2616055</td>
<td>3990325</td>
<td>0</td>
</tr>
<tr>
<td>3990325</td>
<td>6086529</td>
<td>0</td>
</tr>
<tr>
<td>6086529</td>
<td>9283913</td>
<td>0</td>
</tr>
<tr>
<td>9283913</td>
<td>14160950</td>
<td>0</td>
</tr>
<tr>
<td>14160950</td>
<td>21600000</td>
<td>0</td>
</tr>
<tr>
<td>21600000</td>
<td>Infinity</td>
<td>0</td>
</tr>
</tbody>
</table>
REQUEST DATA, or COLLECT AGGREGATE UNIT OF WORK DATA clauses when creating or altering the objects. For work classes, histograms are also collected if you apply a COLLECT AGGREGATE ACTIVITY DATA work action to the work class. The following histograms are available:

- Non-nested coordinator activity lifetime, when you specify AGGREGATE ACTIVITY DATA BASE or AGGREGATE ACTIVITY DATA EXTENDED for a service subclass, for a workload, or for a work action applied to a work class

- Non-nested coordinator activity execution time, when you specify AGGREGATE ACTIVITY DATA BASE or AGGREGATE ACTIVITY DATA EXTENDED for a service subclass, for a workload, or for a work action applied to a work class

- Non-nested coordinator activity queue time, when you specify AGGREGATE ACTIVITY DATA BASE or AGGREGATE ACTIVITY DATA EXTENDED for a service subclass, for a workload, or for a work action applied to a work class

- Request execution time, when you specify AGGREGATE REQUEST DATA BASE for a service subclass; this histogram does not apply to workloads or work classes

- Non-nested activity interarrival time histogram, when you specify AGGREGATE ACTIVITY DATA EXTENDED for a service subclass, for a workload, or for a work action applied to a work class

- Non-nested DML activity estimated cost, when you specify AGGREGATE ACTIVITY DATA EXTENDED for a service subclass, for a workload, or for a work action applied to a work class

- Unit of work lifetime, when you specify AGGREGATE UNIT OF WORK DATA BASE for a service class

All activity-related histograms contain information about activities that are completed, are cancelled, or are rejected.

**Histogram templates**

You can optionally specify a histogram template that is used to determine what a particular histogram looks like, including the high bin value. A histogram template is a *unitless* object, meaning that there is no predefined measurement unit assigned to it. A measurement unit, which depends on the context in which the histogram template is used, is assigned to the histogram when a service class, workload, or work action is created or altered. Time-based histograms, such as those specified with the ACTIVITY LIFETIME HISTOGRAM clause as an example, use millisecond units, while cost-based histograms, such as those specified with the ACTIVITY ESTIMATED COST HISTOGRAM clause as an example, use timeron units.

You can create a histogram template by using the CREATE HISTOGRAM TEMPLATE statement, specifying the maximum high bin value. All other bin values are automatically defined as exponentially increasing values that approach the high bin value. For example, to create a histogram template with a high bin value of 3 000 000, issue a statement such as the following one:

CREATE HISTOGRAM TEMPLATE TEMPLATE1 HIGH BIN VALUE 3000000

This statement creates a histogram template with the following bin values:

<table>
<thead>
<tr>
<th>Low Bin</th>
<th>High Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>
You apply a histogram template by using the appropriate HISTOGRAM TEMPLATE keyword when creating or altering service subclasses, workloads, or work actions. If you do not specify a histogram template, the default template, SYSDEFAULTHISTOGRAM, is used. If you do not enable AGGREGATE ACTIVITY DATA collection for an object, the histogram template is ignored.

For example, to use the TEMPLATE1 histogram template for the existing activity lifetime histogram of service subclass MYSUBCLASS under the service superclass MYSUPERCLASS, issue the following statement:

```
ALTER SERVICE CLASS MYSUBCLASS UNDER MYSUPERCLASS
ACTIVITY LIFETIME HISTOGRAM TEMPLATE TEMPLATE1
```

After you commit the ALTER SERVICE CLASS statement, the activity lifetime histogram that is collected for the MYSUBCLASS service subclass has high bin values that are determined by the TEMPLATE1 histogram template instead of by the SYSDEFAULTHISTOGRAM histogram template.

If you change a service class or a workload to use a different histogram template or change a histogram template, the change does not take effect until a statistics reset occurs.

You can drop a histogram template by using the DROP HISTOGRAM TEMPLATE statement.

You can view the histogram templates by querying the SYSCAT.HISTOGRAMTEMPLATES view and view the corresponding histogram
template high bin values by querying the SYSCAT.HISTOGRAMTEMPLATEBINS view. The low bin value is always 0 for the first bin; for any other bins, the low bin value is the high bin value from the preceding bin.

Example

The following example creates a table function to compute the CoordActLifetime, CoordActExecTime, CoordActQueueTime, or CoordActEstCost histogram for a service superclass as a whole by summing across the subclasses. Summing across subclasses is useful when activities are remapped to different service subclasses under the same service superclass during execution, as can occur under a priority aging scenario where service class tiers and specialized thresholds are used to control resources for activities dynamically. This example does not apply to the CoordActInterArrivalTime histogram because the weighted averages computed do not account for the fact that the CoordActInterArrivalTime histogram of a subclass measures the time between the arrival of a query in that subclass and the next query but the CoordActInterArrivalTime histogram of a superclass measures the time between the arrival of a query in any of its subclasses and the next query.

CONNECT TO SAMPLE

DROP FUNCTION histsuper

CREATE FUNCTION histsuper(superclass varchar(128),
                            histogram_type varchar(24))
RETURNS TABLE (statistics_timestamp timestamp,
               bin_top integer,
               number_in_bin integer,
               graph varchar(60))
LANGUAGE SQL
READS SQL DATA
NO EXTERNAL ACTION
DETERMINISTIC
RETURN WITH HISTOGRAMS AS
  (SELECT HISTOGRAM_TYPE,
       substr(PARENTSERVICECLASSNAME,1,26) as SUPERCLASS,
       STATISTICS_TIMESTAMP,
       TOP as BIN_TOP,
       sum(NUMBER_IN_BIN) as NUMBER_IN_BIN
    FROM HISTOGRAMBIN_DB2STATISTICS H,
         SYSCAT.SERVICECLASSES S
    WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID
    AND PARENTSERVICECLASSNAME = histsuper.superclass
    AND HISTOGRAM_TYPE = histsuper.histogram_type
    AND HISTOGRAM_TYPE IN ('CoordActLifetime', 'CoordActExecTime',
                            'CoordActQueueTime', 'CoordActEstCost')
    GROUP BY HISTOGRAM_TYPE, PARENTSERVICECLASSNAME, STATISTICS_TIMESTAMP, TOP)
SELECT STATISTICS_TIMESTAMP,
     BIN_TOP,
     NUMBER_IN_BIN,
     substr(repeat('#', cast(NUMBER_IN_BIN * 60 /
                     (SELECT CASE WHEN MAX(NUMBER_IN_BIN) = 0 THEN 1
                                   ELSE MAX(NUMBER_IN_BIN) END FROM HISTOGRAMS) AS INTEGER)),1,60)
     AS GRAPH FROM HISTOGRAMS

CONNECT RESET

The output looks as follows:

<table>
<thead>
<tr>
<th>STATISTICS_TIMESTAMP</th>
<th>BIN_TOP</th>
<th>NUMBER_IN_BIN</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-11-06 14:47:08.833188</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2008-11-06 14:47:08.833188</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2008-11-06 14:47:08.833188</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2008-11-06 14:47:08.833188</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2008-11-06 14:47:08.833188</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2008-11-06 14:47:08.833188</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2008-11-06 14:47:08.833188</td>
<td>12</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
Creating a histogram template

Use the CREATE HISTOGRAM TEMPLATE statement to create a histogram template. Histogram templates are used by service subclasses and work actions to define the bin values for the statistics that are maintained using histograms.

Before you begin

To create a histogram template, you require WLMADM or DBADM authority.

See the following topics for more information about prerequisites:
- “Workload management DDL statements” on page 18
- Appendix A, “General naming rules,” on page 533

About this task

Some DB2 service subclass, work class activity, and request statistics are collected using histograms. All histograms have a set number of bins, and each bin represents a range in which activities or requests are counted. The type of units used for the bins depends on the type of histogram that you create. The histogram template describes the high value of the second-to-last bin in the histogram, which affects the values of all of the bins in the histogram. For more information about histograms, see “Histograms in workload management” on page 258.

Procedure

To create a histogram template:

1. Issue the CREATE HISTOGRAM TEMPLATE statement, specifying the name of the histogram template that you want to create and a value for the HIGH BIN VALUE keyword to set the top value for the second-to-last bin.
2. Commit your changes. When you commit your changes, the histogram is added to the SYSCAT.HISTOGRAMTEMPLATES view and the bins are added to the SYSCAT.HISTOGRAMTEMPLATEBINS view.

**Altering a histogram template**

Use the ALTER HISTOGRAM TEMPLATE statement to alter an existing histogram template. Histogram templates are used by service subclasses and work actions to define the bin values for the statistics that are maintained using histograms.

**Before you begin**

You require WLMADM or DBADM authority to alter a histogram template.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

**About this task**

Some DB2 service subclass, work class activity, and request statistics are collected using histograms. All histograms have a set number of bins, and each bin represents a range in which activities or requests are counted. The type of units used for the bins depends on the type of histogram that you create. The histogram template describes the high value of the second-to-last bin in the histogram, which affects the values of all of the bins in the histogram. For more information on histograms, see “Histograms in workload management” on page 258.

**Procedure**

To alter a histogram template:

1. Issue the ALTER HISTOGRAM TEMPLATE statement, specifying the name of the histogram template that you want to alter and a value for the HIGH BIN VALUE parameter to alter the top value for the second-to-last bin.

2. Commit your changes. When you commit your changes the high bin value for the histogram is updated in the SYSCAT.HISTOGRAMTEMPLATEBINS view. The change does not take effect until the next time the workload management statistics are reset. See “Resetting statistics on DB2 workload management objects” on page 269 for more information.

3. Optional: Run the WLM_COLLECT_STATS stored procedure to collect and reset the statistics so that the new histogram template is used immediately.

**Dropping a histogram template**

You can drop a histogram template if you no longer require it.

**Before you begin**

To drop a histogram template, you require WLMADM or DBADM authority.

See “Workload management DDL statements” on page 18 for more information about prerequisites.

You cannot drop the SYSDEFAULTHISTOGRAM histogram template.

You cannot drop a histogram template if it is being referenced by a service subclass, work action, or workload. You can view the service subclasses and work actions that reference a histogram template by querying the
SYSCAT.HISTOGRAMTEMPLATESUSE view.

**Procedure**

To drop a histogram template:
1. Use the DROP HISTOGRAM TEMPLATE statement.
2. Commit your changes. When you commit your changes the histogram is removed from the SYSCAT.HISTOGRAMTEMPLATES view, and its bins are removed from the SYSCAT.HISTOGRAMTEMPLATEBINS view.

**Example: Computing averages and a standard deviation from histograms in a DB2 workload management configuration**

One use for histograms is for obtaining the standard deviation for activity lifetimes. The example in this topic shows how bins are used for the calculation of this statistic.

A calculation of the average lifetime for each activity is a useful piece of information. However, the average alone does not accurately describe the user experience. If the variability in activity lifetime is large, the users whom you are supporting might see queries run fast at some times (which is fine) and slow at others (which might not be acceptable). When you define a goal for activity lifetimes, not only is the average lifetime of the activities important but also the standard deviation of the activity lifetime. You need to both understand and control variability to ensure that your users actually experience the observed average.

In a DB2 workload management configuration, statistics are collected on each member. The following example shows how to obtain the average activity lifetime for a single member.

Suppose that you have a single-member environment and histogram with the following bins. There are more bins in the real histograms, but this example is limited to eight bins to make the example simpler.

<table>
<thead>
<tr>
<th>Bin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin 1</td>
<td>0 to 2 seconds</td>
</tr>
<tr>
<td>Bin 2</td>
<td>2 to 4 seconds</td>
</tr>
<tr>
<td>Bin 3</td>
<td>4 to 8 seconds</td>
</tr>
<tr>
<td>Bin 4</td>
<td>8 to 16 seconds</td>
</tr>
<tr>
<td>Bin 5</td>
<td>16 to 32 seconds</td>
</tr>
<tr>
<td>Bin 6</td>
<td>32 to 64 seconds</td>
</tr>
<tr>
<td>Bin 7</td>
<td>64 to 128 seconds</td>
</tr>
<tr>
<td>Bin 8</td>
<td>128 seconds to infinity</td>
</tr>
</tbody>
</table>

You can compute an approximation of the average by assuming that the average response time for a query that falls into a bin with the range $x$ to $y$ is $(x + y)/2$. You can then multiply this number by the number of queries that fell into the bin, sum across all bins, then divide the sum by the total count. For the preceding example, assume that the average response time for each bin is:

<table>
<thead>
<tr>
<th>Bin</th>
<th>Average Lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin 1</td>
<td>$(0+2)/2 = 1$</td>
</tr>
<tr>
<td>Bin 2</td>
<td>$(2+4)/2 = 3$</td>
</tr>
<tr>
<td>Bin 3</td>
<td>$(4+8)/2 = 6$</td>
</tr>
<tr>
<td>Bin 4</td>
<td>$(8+16)/2 = 12$</td>
</tr>
<tr>
<td>Bin 5</td>
<td>$(16+32)/2 = 24$</td>
</tr>
<tr>
<td>Bin 6</td>
<td>$(32+64)/2 = 48$</td>
</tr>
<tr>
<td>Bin 7</td>
<td>$(64+128)/2 = 96$</td>
</tr>
</tbody>
</table>

Assume that the following histogram was collected during the measurement period:
To calculate average lifetime, bin 8 must be empty. Bin 8 only exists to let you know when you need to change the upper boundary of your range. For this reason, you must specify the upper bound for the range.

You can approximate the average lifetime for member 1 as follows:

\[
\text{average lifetime} = \frac{(20 \times 1 + 30 \times 3 + 80 \times 6 + 10 \times 12 + 5 \times 24 + 3 \times 48 + 2 \times 96)}{150}
\]

\[
= \frac{(20 + 90 + 480 + 120 + 120 + 144 + 192)}{150}
\]

\[
= \frac{1166}{150}
\]

\[
= 7.77 \text{ seconds}
\]

You can approximate the lifetime standard deviation as follows:

\[
\text{Standard deviation} = \sqrt{\frac{(20 \times (1 - 7.77)^2 + 30 \times (3 - 7.77)^2 + \ldots)}{150}}
\]

For multimember database environments, averages and standard deviations can be computed by first computing a combined histogram across all database members by adding the counts of each bin across the database members.

For example, assume that the database has two members, the histogram bin sizes are as described previously, and the histogram has the following data:

<table>
<thead>
<tr>
<th>Database member</th>
<th>Bin 1 count</th>
<th>Bin 2 count</th>
<th>Bin 3 count</th>
<th>Bin 4 count</th>
<th>Bin 5 count</th>
<th>Bin 6 count</th>
<th>Bin 7 count</th>
<th>Bin 8 count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>30</td>
<td>80</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Because the bin sizes are the same across all database members, the overall histogram is easy to compute:

<table>
<thead>
<tr>
<th>Bin 1 count</th>
<th>Bin 2 count</th>
<th>Bin 3 count</th>
<th>Bin 4 count</th>
<th>Bin 5 count</th>
<th>Bin 6 count</th>
<th>Bin 7 count</th>
<th>Bin 8 count</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>35</td>
<td>100</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

From the combined histogram, you can calculate the overall lifetime average and standard deviation in a similar way to how they were computed for a single-member environment:

\[
\text{Average lifetime} = \frac{(21 \times 1 + 35 \times 3 + 100 \times 6 + 9 \times 12 + 3 \times 48 + 2 \times 96)}{200}
\]

\[
= \frac{(21 + 105 + 600 + 216 + 144 + 192)}{200}
\]

\[
= \frac{1638}{200}
\]

\[
= 8.19 \text{ seconds}
\]

\[
\text{Standard deviation} = \sqrt{\frac{(21 \times (1 - 8.19)^2 + 35 \times (3 - 7.77)^2 + \ldots)}{200}}
\]

**Historical analysis tool**

Your DB2 data server installation includes a pair of Perl scripts as a sample that generate information about which tables, indexes, and columns have or have not been accessed by performing historical analysis.

These scripts provide historical analysis functionality by using information captured by the workload management activities event monitor. The workload management historical analysis tool was written in Perl; you can use these scripts as is or you can modify them to produce additional historical analysis reports to suit your needs.

The workload management historical analysis tool consists of two scripts, which can be found in the samples/perl path of your installation directory:
Collecting workload management statistics using a statistics event monitor

Statistics for DB2 workload management objects can be sent to a statistics event monitor for historical analysis.

About this task

You can use statistics to understand the behavior of your system over time (for example, what is the average lifetime of activities, how much time do activities spend queued, what is the distribution of large compared to small activities, and so on), set thresholds (for example, find the upper boundary for concurrent activities), and detect problems (for example, detect whether the average lifetime that users are experiencing is higher than normal). See “Statistics for DB2 workload management objects” on page 246 for a description of which statistics are collected for each DB2 workload management object.

You can automatically send workload management statistics to an event monitor on a fixed interval of time, or you can manually send statistics to an event monitor at any point in time.

Procedure

To automatically collect workload management statistics on a fixed time interval:

1. Use the CREATE EVENT MONITOR statement to create a STATISTICS event monitor. For example, you could issue the following statement:
   
   ```sql
   CREATE EVENT MONITOR STATS1 FOR STATISTICS WRITE TO TABLE
   ```

2. Use the COMMIT statement to commit your changes.

3. Use the SET EVENT MONITOR STATE statement to activate the event monitor. Instead of using the SET EVENT MONITOR STATE statement, you can use the AUTOSTART default for the STATISTICS event monitor to have it activated the next time that the database is activated. If you want to define multiple STATISTICS event monitors, you should not use the AUTOSTART option.

4. Use the COMMIT statement to commit your changes.

5. Optional: Enable the collection of additional statistics. By default, only a minimal set of statistics is collected for each DB2 workload management object. See “Statistics for DB2 workload management objects” on page 246 for details on which statistics are collected by default for each object. Specify the collection of aggregate activity data for service subclasses, workloads, and work classes using the COLLECT AGGREGATE ACTIVITY DATA keyword on the ALTER SERVICE CLASS and ALTER WORK ACTION SET statements. Specify the collection of aggregate request data for service subclasses using the COLLECT AGGREGATE REQUEST DATA keyword on the ALTER SERVICE CLASS statement. COMMIT any changes.
6. Specify a collection interval by updating the database configuration parameter \texttt{wlm\_collect\_int}. The \texttt{wlm\_collect\_int} parameter specifies an interval of time in minutes. Every interval, the copy of the workload management statistics for all DB2 workload management objects is written to the active statistics event monitor and the statistics are reset. In a multimember database environment, the \texttt{wlm\_collect\_int} parameter must be updated on the catalog member. This parameter can be updated dynamically. For example:

\begin{verbatim}
CONNECT TO database alias
UPDATE DATABASE CONFIGURATION USING WLM_COLLECT_INT 5 IMMEDIATE
\end{verbatim}

\textbf{Results}

After you perform the preceding steps, workload management statistics are written to the statistics event monitor every \texttt{wlm\_collect\_int} minutes. Each record written to the statistics event monitor has a \texttt{STATISTICS\_TIMESTAMP} value and a \texttt{LAST\_WLM\_RESET} value. The interval of time from \texttt{LAST\_WLM\_RESET} to \texttt{STATISTICS\_TIMESTAMP} defines the collection interval (that is, interval of time over which the statistics in that record were collected).

Collections occur at the specified interval times as measured relative to Sunday at 00:00:00. When the catalog member becomes active, the next collection will occur at the start of the next scheduled interval relative to this fixed time. The scheduled interval is not relative to the catalog member activation time. If a member is not active at the time of collection, no statistics are gathered for that member. For example, if the interval value was set to 60 and the catalog member was activated on 09:24 AM on Sunday, then the collections would be scheduled to occur each hour on the hour. Therefore, the next collection will occur at 10:00 AM. If the member is not active at 10:00 AM, then no statistics will be gathered for that member.

If the \texttt{wlm\_collect\_int} parameter is set to a nonzero value and there is no active statistics event monitor, the workload management statistics are still reset every \texttt{wlm\_collect\_int} minutes, but statistics are not collected. The data will be lost. For this reason, it is not recommended that you specify a nonzero \texttt{wlm\_collect\_int} value without activating a statistics event monitor.

If the \texttt{wlm\_collect\_int} parameter is set to 0 (the default) statistics are not sent to the statistics event monitor automatically. You can manually send statistics to the statistics event monitor for later historical analysis by using the \texttt{WLM\_COLLECT\_STATS} stored procedure. When this procedure is invoked, it performs the same actions that occur with an automatic statistics collection interval. That is, the statistics are sent to the statistics event monitor and the statistics are reset. If there is no active statistics event monitor, the values are reset, but data is not collected. If you only want to reset statistics, you can invoke the \texttt{WLM\_COLLECT\_STATS} procedure while there is no active statistics event monitor.

Manual collection of statistics does not interfere with the automatic collection of statistics. For example, assume that you have \texttt{wlm\_collect\_int} set to 60. Statistics are sent to the statistics event monitor every hour. Now assume that the last time the statistics were collected was 05:00 AM. You can invoke the \texttt{WLM\_COLLECT\_STATS} procedure at 05:55 AM, which sends the values of the statistics to the event monitor and resets the statistics. The next automatic statistics collection still occurs at 06:00 AM, one hour after the last automated collection. The collection interval is not affected by any manual collection and resetting of statistics that occurs during the interval.

\textbf{Notes}:
The DB2 workload management statistics table functions report the current values of the statistics. If you have automatic workload management statistics collection enabled, these values are reset periodically on the interval defined by the `wlm_collect_int` database configuration parameter. When looking at the statistics reported by the table functions, you should always consider the `LAST_RESET` column. This column indicates the last time the statistics were reset. If the time interval between the last reset time to the current time is not sufficiently large, there may not be enough data to draw any meaningful conclusions.

If you are using automatic collection of workload management statistics, you need to prune your event monitor files or tables periodically. The event monitor does not automatically prune the data that is collected, and the automatic collection will fill your files or tables over time.

When a database is deactivated, the statistics are reset. Deactivating the database does not send statistics to the statistics event monitor. If you do not want to lose the statistics accumulated since the last collection because of a deactivation, you should manually invoke the WLM_COLLECT_STATS procedure before deactivating the database.

The WLM_COLLECT_STATS procedure resets statistics differently than the `RESET MONITOR` command. The `RESET MONITOR` command resets the values of snapshot monitor elements by storing their present values. After the `RESET MONITOR` command has been issued, snapshot processing reports the delta between these values and the current values. In contrast, the reset caused by the WLM_COLLECT_STATS procedure does not store any values, but instead resets all of the statistics counters themselves for each applicable DB2 workload management object.

Also, with the `RESET MONITOR` command, each process (attachment) has its own private view of the monitor data. If one user performs a reset, other users are unaffected. By contrast, a reset of the workload management statistics applies to all users.

### Resetting statistics on DB2 workload management objects

This topic describes how to reset statistics for DB2 workload management objects.

Note that resetting statistics applies only to DB2 workload management statistics; metrics reported by monitoring interfaces will be collected, but not reset.

Four events will reset the statistics stored for each DB2 workload management object. (For a description of the statistics maintained for each object, see “Statistics for DB2 workload management objects” on page 246.)

- The WLM_COLLECT_STATS stored procedure is invoked. See “Collecting workload management statistics using a statistics event monitor” on page 267 for details.
- The automatic DB2 workload management statistics collection and reset process controlled by the `wlm_collect_int` database configuration parameter causes a collection and reset. See “Collecting workload management statistics using a statistics event monitor” on page 267 for details.
- The database is reactivated. Every time the database is activated on a member, the statistics for all DB2 workload management objects on that member are reset.
- The object for which the statistics are maintained is modified and the change is committed. For example if a service subclass is altered, when the `ALTER` statement is committed, the statistics for that service subclass are reset.
You can determine the last time the statistics were reset for a given DB2 workload management object using the statistics table functions and looking at timestamp in the LAST_RESET column. For example, to see the last time the statistics were reset for the service subclass SYSDEFAULTSUBCLASS under the SYSDEFAULTUSERCLASS service superclass, you could issue a query such as:

```
SELECT LAST_RESET
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS( 'SYSDEFAULTUSERCLASS',
 'SYSDEFAULTSUBCLASS', -2)) AS T
```

All statistics table functions return the statistics that accumulated since the last time that the statistics were reset. A statistics reset occurs when a database is activated or reactivated, when you alter a DB2 workload management object (only the statistics for that object are reset), and when you call the WLM_COLLECT_STATS stored procedure. Statistics are also reset automatically according to the time period defined by the `wlm_collect_int` database configuration parameter, if you set this parameter to a nonzero value.

The period of time specified by `wlm_collect_int` is unaffected by a statistics reset that occurs during the interval specified by the configuration parameter. For example, if you run the WLM_COLLECT_STATS table function 5 minutes after the start of a 20-minute interval specified by `wlm_collect_int`, the interval still expires 15 minutes later. The statistics collection and reset that occur do not delay the occurrence of the next statistics collection and reset by 5 minutes.

If you change a service class or a workload to use a different histogram template or change a histogram template, the change does not take effect until a statistics reset occurs.

If you invoke the WLM_COLLECT_STATS table function to collect and reset statistics at the same time that another collection and reset is in progress (for example, if the invocation of the table function overlaps with the periodic collection and reset interval caused by `wlm_collect_int` or if another user invokes WLM_COLLECT_STATS at the same time), the collection and reset request from WLM_COLLECT_STATS is ignored, and warning SQL1632W is returned.

---

**Monitoring metrics for DB2 workload management**

Monitoring metrics provide data about the health of and query performance on your DB2 data server, which can then be used as input to a 3rd party tool or in conjunction with additional scripting you provide to analyze the metrics returned.

Metrics are maintained for a number of DB2 database objects. These metrics reside in memory and can be viewed in real-time using DB2 monitoring metrics table functions, or the metrics can be collected and sent to an event monitor where they can be viewed later for historical analysis.

**Monitoring metrics for activities**

You can obtain monitoring metrics for activities using:

- The activities event monitor (the ACTIVITYMETRICS table or the DETAILS_XML column of the ACTIVITY table)
- The MON_GET_ACTIVITY_DETAILS table function

Monitoring metrics for activities are controlled by the `mon_act_metrics` database configuration parameter and the COLLECT ACTIVITY METRICS clause on a workload. Metrics will be collected for an activity, if the database configuration
parameter is set to a value other than NONE or if the activity is submitted by a connection that is associated with a workload which has a COLLECT ACTIVITY METRICS setting other than NONE.

You can use workload-level controls to achieve better monitoring granularity, if you do not want to collect metrics for all activities. If activity metrics collection is enabled at the database level (enabled by default), then metrics are collected for all activities, regardless of the setting at the workload level.

See the monitoring documentation for more details.

**System-level monitoring metrics**

You can obtain system-level monitoring metrics aggregated by service classes and workloads using:

- The statistics event monitor (the `details_xml` and `metrics` monitor elements in the `EVENT_SCSTATS` and `EVENT_WLSTATS` logical data groups or individual monitor elements in the `EVENT_SCMETRICS` and `EVENT_WLMETRICS` logical data groups)
- The `MON_GET_SERVICE_SUBCLASS`, `MON_GET_SERVICE_SUBCLASSDETAILS`, `MON_GET_WORKLOAD` and `MON_GET_WORKLOADDETAILS` table functions

Monitoring metrics for requests to the data server, including those requests that are part of an activity, are controlled by the `mon_req_metrics` database configuration parameter and the COLLECT REQUEST METRICS clause on a service superclass. Metrics will be collected for a request, if the database configuration parameter is set to a value other than NONE or if the request is submitted by a connection that mapped to a subclass under a superclass which has a COLLECT REQUEST METRICS setting other than NONE.

You can use service superclass-level controls to achieve better monitoring granularity, if you do not want to collect metrics for all requests. If request metrics collection is enabled at the database level (enabled by default), then metrics are collected for all requests, regardless of the setting at the service superclass level.

See the monitoring documentation for more details.

**Workload management table functions and snapshot monitor integration**

You can use DB2 workload management table functions with the snapshot monitor table functions when performing problem determination or performance tuning.

The DB2 workload management table functions and the snapshot monitor table functions share the following fields. You can perform joins on these fields to derive data that you need to perform diagnostic and performance-tuning activities. Note that, unlike the snapshot table functions, the WLM table functions do not get their information from the snapshot monitor, so that the information available in the WLM table functions is not available from the snapshot monitor.
Table 67. Fields shared between the DB2 workload management and snapshot monitor table functions

<table>
<thead>
<tr>
<th>Workload management table function field</th>
<th>Snapshot monitor table function field</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_tid</td>
<td>agent_pid</td>
</tr>
<tr>
<td>application_handle</td>
<td>agent_id</td>
</tr>
<tr>
<td></td>
<td>agent_id_holding_lock</td>
</tr>
<tr>
<td>session_auth_id</td>
<td>session_auth_id</td>
</tr>
<tr>
<td>member</td>
<td>node_number</td>
</tr>
<tr>
<td>utility_id</td>
<td>utility_id</td>
</tr>
<tr>
<td>workload_id</td>
<td>workload_id</td>
</tr>
</tbody>
</table>

As an example of a reason to use a join between different table functions, assume that you want to obtain basic information about all of the utilities running in the BATCH service superclass. You might issue the following query:

```sql
SELECT SUBSTR(UTILITY_TYPE,1,4) TYPE, 
  UTILITY_PRIORITY PRIORITY, 
  SUBSTR(UTILITY_DESCRIPTION,1,12) DESCRIPTION, 
  SUBSTR(UTILITY_DBNAME,1,8) DBNAME, 
  UTILITY_STATE STATE, 
  SUBSTR(UTILITY_INVOKER_TYPE,1,7) INVOKER, 
  SUBSTR(CHAR(WLM.MEMBER),1,4) MEMB, 
  SUBSTR(CLASSES.PARENTSERVICECLASSNAME,1,19) SUPERCLASS_NAME, 
  SUBSTR(CLASSES.SERVICECLASSNAME,1,18) SUBCLASS_NAME 
FROM SYSIBMADM.SNAPUTIL SNAP, 
  TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(CAST(NULL AS BIGINT), -2)) WLM, 
  SYSCAT.SERVICECLASSES CLASSES 
WHERE SNAP.UTILITY_ID = WLM.UTILITY_ID 
  AND WLM.SERVICE_CLASS_ID = CLASSES.SERVICECLASSID 
  AND CLASSES.SERVICECLASSNAME = 'SYSDEFAULTSUBCLASS' 
  AND CLASSES.PARENTSERVICECLASSNAME = 'BATCH' 
ORDER BY WLM.MEMBER
```

The output might resemble the following output:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>PRIORITY</th>
<th>DESCRIPTION</th>
<th>DBNAME</th>
<th>STATE</th>
<th>INVOKER</th>
<th>MEMB</th>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>- OFFLINE</td>
<td>LOAD SAMPLE EXECUTE USER 1</td>
<td>BATCH</td>
<td>SYSDEFAULTSUBCLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>- OFFLINE</td>
<td>LOAD SAMPLE EXECUTE USER 1</td>
<td>BATCH</td>
<td>SYSDEFAULTSUBCLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>- OFFLINE</td>
<td>LOAD SAMPLE EXECUTE USER 1</td>
<td>BATCH</td>
<td>SYSDEFAULTSUBCLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>- OFFLINE</td>
<td>LOAD SAMPLE EXECUTE USER 2</td>
<td>BATCH</td>
<td>SYSDEFAULTSUBCLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>- OFFLINE</td>
<td>LOAD SAMPLE EXECUTE USER 2</td>
<td>BATCH</td>
<td>SYSDEFAULTSUBCLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>- OFFLINE</td>
<td>LOAD SAMPLE EXECUTE USER 3</td>
<td>BATCH</td>
<td>SYSDEFAULTSUBCLASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Monitoring threshold violations**

When a DB2 workload manager threshold is violated, a threshold violation record is written to the active THRESHOLD VIOLATIONS event monitor, if one exists.

**About this task**

The threshold violation record includes the following information:

- A description of the threshold that was violated (the identifier, maximum value, and so on).
- An identification of the activity that violated the threshold, including the identifier of the application that submitted the activity, the unique activity identifier, and the unit of work identifier.
• The time that the threshold was violated.
• The action that was taken. The action indicates whether the activity that violated the threshold was permitted to continue or was stopped. If the activity was stopped, the application that submitted the activity will have received an SQL4712N error.

When a threshold violation occurs for a threshold that has a REMAP ACTIVITY action defined for it, a threshold violation record is optional. Whether or not a threshold violation record is recorded is determined by the NO EVENT MONITOR RECORD or LOG EVENT MONITOR RECORD clause of your CREATE THRESHOLD statement.

You can optionally have detailed activity information (including statement text) written to an active activities event monitor if the threshold violation is caused by an activity. The activity information is written when the activity completes, not when the threshold is violated. Specify that activity information should be collected when a threshold is violated by using the COLLECT ACTIVITY DATA keyword on either the CREATE or ALTER threshold or work action set statements.

**Procedure**

To monitor threshold violations:

1. Use the CREATE EVENT MONITOR statement to create an event monitor of type THRESHOLD VIOLATIONS. For example:
   ```sql
   CREATE EVENT MONITOR VIOLATIONS FOR THRESHOLD VIOLATIONS WRITE TO TABLE
   ```
2. Use the COMMIT statement to commit your changes.
3. Use the SET EVENT MONITOR STATE statement to activate the event monitor. Instead of using the SET EVENT MONITOR STATE statement, you can use the AUTOSTART default for the THRESHOLD VIOLATIONS event monitor to have it activated the next time that the database is activated. If you want to define multiple THRESHOLD VIOLATIONS event monitors, you should not use the AUTOSTART option.
4. Use the COMMIT statement to commit your changes.

   **Note:** If you create any thresholds, you should create and activate a threshold violations event monitor so you can monitor any threshold violations that occur. A threshold violations event monitor does not have any impact unless thresholds are violated.

**Example**

This example shows how you can determine what remappings of a particular activity occurred as the result of a threshold violation that included a REMAP ACTIVITY action. To find the activities that were remapped, use a statement like the following:

```sql
SELECT VARCHAR(APPL_ID, 30) AS APPLID,
      UOW_ID,
      ACTIVITY_ID,
      VARCHAR(T.PARENTSERVICECLASSNAME, 20) AS SERVICE_SUPERCLASS,
      VARCHAR(T.SERVICECLASSNAME, 20) AS FROM_SERVICE_SUBCLASS,
      VARCHAR(S.SERVICECLASSNAME, 20) AS TO_SERVICE_SUBCLASS
FROM THRESHOLDVIOLATIONS_TH1,
     SYSCAT.SERVICECLASSES AS T,
     SYSCAT.SERVICECLASSES AS S
WHERE SOURCE_SERVICE_CLASS_ID = T.SERVICECLASSID AND
```

Chapter 4. Monitoring and intervention 273
DESTINATION_SERVICE_CLASS_ID = S.SERVICECLASSID AND
THRESHOLD_ACTION = 'REMAP'
ORDER BY APPLID, ACTIVITY_ID, UOW_ID, TIME_OF_VIOLATION ASC;

In this example, two remappings occurred for the activity submitted by the application with the ID *N0.swalkty.080613140844 which is identified by activity ID 1 and unit of work (UOW) ID 1:

<table>
<thead>
<tr>
<th>APPLID</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>SERVICE_SUPERCLASS</th>
<th>FROM_SERVICE_SUBCLASS</th>
<th>TO_SERVICE_SUBCLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>*N0.swalkty.080613140844</td>
<td>1</td>
<td>1 WORK</td>
<td>HIGH</td>
<td>MED</td>
<td>LOW</td>
</tr>
</tbody>
</table>

The output is ordered by the time of threshold violation and shows that the activity was remapped twice after it started executing. Although not shown in the output, the initial service subclass the activity was mapped to is likely a high priority service subclass, typical of a three-tiered configuration that permits shorter running queries to complete more quickly. Because the activity did not complete quickly enough in the high priority service subclass, it violated a threshold and was remapped to a medium priority service subclass, and then remapped again to a low priority service subclass after a second threshold violation later on.

How to generate email notifications for threshold violations

Using the method described here, you can generate email notifications when a DB2 workload manager (WLM) threshold is violated.

Before you begin

In order to implement this email notification approach, you must have DB2 Version 9.7 or higher installed. The SMTP support used here was provided since DB2 V9.7.

About this task

Upon completion of this task, email notifications are sent if WLM threshold violations occur during the 10 minutes since the threshold notification procedure was last run. The DB2 Administrative Task Scheduler is used to schedule the threshold notification procedure to run every 10 minutes in this example.

Procedure

1. Update the smtp_server database configuration parameter by issuing the following command:

   UPDATE DB CONFIG USING SMTP_SERVER smtp_server_name

2. Create a write-to-table event monitor for threshold violations and write violations to the TEST.THRESHOLDVIOLATIONS_T table by issuing the following statement:

   CREATE EVENT MONITOR T FOR THRESHOLD VIOLATIONS WRITE TO TABLE
   THRESHOLDVIOLATIONS( TABLE TEST.THRESHOLDVIOLATIONS_T )

3. Activate the write-to-table event monitor T for threshold violations by issuing the following statement:

   SET EVENT MONITOR T STATE 1

4. Create a control table to track the last threshold for which an alert was generated by issuing the following statement:

   CREATE TABLE TEST.THRESHOLD_NOTIFY_CONTROL( LAST_NOTIFICATION_TIMESTAMP )
5. Create a stored threshold notification procedure to generate threshold violation messages. The following example procedure iterates over the threshold violations table and builds a report listing all threshold violations that have occurred since the last time the procedure was invoked. The report is emailed using the DB2 SMTP procedures.

```sql
CREATE PROCEDURE TEST.NOTIFY_ON_THRESHOLD_VIOLATION()
LANGUAGE SQL
BEGIN

DECLARE NEWEST_VIOLATION TIMESTAMP;
DECLARE LAST_VIOLATION_SEEN TIMESTAMP;
DECLARE NOT_FOUND INTEGER DEFAULT 0;
DECLARE SENDER VARCHAR(128);
DECLARE RECIPIENTS VARCHAR(128);
DECLARE MESSAGE VARCHAR(8192);
DECLARE SUBJECT VARCHAR(128);
DECLARE THRESHOLDID BIGINT;
DECLARE APPL_ID VARCHAR(64);
DECLARE THRESHOLD.Predicate VARCHAR(64);
DECLARE TIME_OF_VIOLATION TIMESTAMP;

DECLARE C1 CURSOR FOR SELECT MAX(TIME_OF_VIOLATION)
FROM TEST.THRESHOLDVIOLATIONS_T;
DECLARE C2 CURSOR FOR SELECT LAST_NOTIFICATION
FROM TEST.THRESHOLD_NOTIFY_CONTROL;
DECLARE C3 CURSOR FOR SELECT THRESHOLD.Predicate, THRESHOLDID, TIME_OF_VIOLATION, APPL_ID
FROM TEST.THRESHOLDVIOLATIONS_T
WHERE LAST_VIOLATION_SEEN IS NULL OR TIME_OF_VIOLATION > LAST_VIOLATION_SEEN;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET NOT_FOUND = 1;

OPEN C1;
FETCH C1 INTO NEWEST_VIOLATION;
CLOSE C1;

IF ( NOT_FOUND = 0 ) THEN

OPEN C2;
FETCH C2 INTO LAST_VIOLATION_SEEN;
CLOSE C2;

IF ( NOT_FOUND = 1 ) THEN
SET LAST_VIOLATION_SEEN = NULL;
END IF;

IF ( NOT_FOUND = 1 OR NEWEST_VIOLATION > LAST_VIOLATION_SEEN ) THEN
DELETE FROM TEST.THRESHOLD_NOTIFY_CONTROL;
INSERT INTO TEST.THRESHOLD_NOTIFY_CONTROL VALUES
( NEWEST_VIOLATION );

SET SENDER = '<sender email address>';  
SET RECIPIENTS = '<receiver email address>';  
SET SUBJECT = 'New WLM Threshold Violations' ;
SET NOT_FOUND = 0;
SET MESSAGE = '';

OPEN C3;
FETCH C3 INTO THRESHOLD.Predicate, THRESHOLDID, TIME_OF_VIOLATION, APPL_ID;

WHILE ( NOT_FOUND = 0 ) DO

```

Chapter 4. Monitoring and intervention 275
6. Enable the DB2 Administrative Task Scheduler by running the following command:
   
   ```sql
   db2set DB2_ATS_ENABLE=YES
   ```

7. Schedule the threshold notification procedure to execute every 10 minutes. To schedule the procedure, you must have execute privileges on the procedure. The following is an example of how this can be done:

   ```sql
   CALL SYSPROC.ADMIN_TASK_ADD('CHECK THRESHOLD VIOLATIONS EVERY 10 MINUTES',
      NULL, NULL, NULL, '0-59/10 * * * *',
      'TEST', 'NOTIFY_ON_THRESHOLD_VIOLATION',
      NULL, NULL, NULL )
   ```

**Results**

An email is sent whenever a WLM threshold violation occurs (with a latency of at most 10 minutes). The email describes all the WLM threshold violations since the last run of the threshold violation procedure scheduled to run every 10 minutes.

**Example**

The following output is an example of the content of an email notification message showing the accumulated new threshold violations since the last run of the threshold violation procedure:

```
Subject: New WLM Threshold Violations

Violation Timestamp = 2010-01-11-10.57.21.000000
Threshold Predicate = CPUTime
Threshold Id = 1
Appl Id = *LOCAL.horton.10011154912
```
Collecting data for individual activities

You can use an ACTIVITIES event monitor to collect data for individual activities that run in your system. The data collected includes items such as statement text and compilation environment, and can be used to investigate and diagnose problems, and as input to other tools (for example, the Design Advisor).

About this task

You can collect information about individual activities for service subclasses, workloads, work classes (through work actions), and threshold violations. You enable activity collection using the COLLECT ACTIVITY DATA keyword of the CREATE and ALTER statements for these DB2 workload management objects. When an activity completes, information about the activity is sent to the active ACTIVITIES event monitor if:

- The activity was submitted by an application that is mapped to a workload for which COLLECT ACTIVITY DATA is specified, or
- The activity runs in a service subclass for which COLLECT ACTIVITY DATA is specified, or
- The activity has a COLLECT ACTIVITY DATA work action applied to it, or
- The activity violates a threshold that was defined with the COLLECT ACTIVITY DATA action

You can also use the WLM_SET_CONN_ENV procedure to turn on activity collection for your own application’s connection before executing the user’s query, then execute the user’s query, then use WLM_SET_CONN_ENV to turn off activity collection for your application’s own connection. Assuming that you have created and activated an activity event monitor, the application could look something like the following:

```sql
CALL WLM_SET_CONN_ENV(cast(NULL as bigint), 'collectactdata>WITHOUT DETAILS</collectactdata>')

... execute user's query ...

CALL WLM_SET_CONN_ENV(cast(NULL as bigint), 'collectactdata>NONE</collectactdata>')
```

The COLLECT ACTIVITY DATA keyword also controls the amount of information that is sent to the ACTIVITIES event monitor. If the keyword specifies WITH DETAILS, statement information (such as statement text) is collected. If the keyword specifies WITH DETAILS AND VALUES, data values are collected as well.

An activity might have multiple COLLECT ACTIVITY DATA keywords applied to it. For example, the activity might run in a service subclass for which COLLECT ACTIVITY DATA is specified, and while executing it might violate a threshold that has the COLLECT ACTIVITY DATA action. In this situation, the activity is only collected once. The COLLECT keyword that specifies the largest amount of...
information to be collected is applied to the activity. For example, if both
COLLECT ACTIVITY DATA WITHOUT DETAILS and COLLECT ACTIVITY DATA
WITH DETAILS are applied to an activity, the activity is collected with detailed
information.

If the ON ALL DATABASE MEMBERS keywords are used with the COLLECT
ACTIVITY DATA clause, an activity record will be captured on each member
where the activity executes in a multimember database environment. Activity event
monitor records are written when the last agent working on the activity at that
member completes execution. Depending on the sequencing of events in a section,
it is possible for agents to start and stop working on an activity at a member
several times, causing multiple activity records to be captured at that member for
the same query. The total work done by the activity on that member is the
aggregate of the metrics for each record that is captured for the activity on the
member.

**Procedure**

To enable collection of activities for a given DB2 workload management object:
1. Use the CREATE EVENT MONITOR statement to create an ACTIVITIES event
   monitor.
2. Use the COMMIT statement to commit your changes.
3. Use the SET EVENT MONITOR STATE statement to activate the event monitor.
   Instead of using the SET EVENT MONITOR STATE statement, you can use the
   AUTOSTART default for the ACTIVITIES event monitor to have it activated the
   next time that the database is activated. If you want to define multiple
   ACTIVITIES event monitors, you should not use the AUTOSTART option.
4. Use the COMMIT statement to commit your changes.
5. Identify the objects for which you want to collect activities by using the ALTER
   SERVICE CLASS, ALTER WORK ACTION SET, ALTER THRESHOLD, or
   ALTER WORKLOAD statement and specify the COLLECT ACTIVITY DATA
   keywords.
6. Use the COMMIT statement to commit your changes.

**Results**

**Note**: Individual activity collection is more expensive than workload management
statistics collection. You should try to set up activity collection to collect as few
activities as possible. For example, if you need to investigate activities submitted
by a specific application, you could isolate that application by creating a workload
or service class specifically for that application, and only enable activity collection
for that workload or service class.

You might not always know in advance that you will want to capture an activity.
For example, you might have a query that is taking a long time to run and you
want to collect information about it for later analysis. In this situation, it is too late
to specify the COLLECT ACTIVITY DATA keyword on the DB2 workload
management objects, because the activity has already entered the system. In this
situation, you can use the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored
procedure. The WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure
sends information about an executing activity to the active ACTIVITIES event
monitor. You identify the activity to be collected using the application handle, unit
of work identifier, and activity identifier. Information about the activity is
immediately be sent to the ACTIVITIES event monitor when the procedure is
invoked: you do not need to wait for the activity to complete.

**Importing activity information into the Design Advisor**

You can import activities collected by an activities event monitor into the Design Advisor to help you make decisions about the database objects accessed by these activities.

**About this task**

Activities imported into the design advisor must have been collected using the COLLECT ACTIVITY DATA WITH DETAILS or COLLECT ACTIVITY DATA WITH DETAILS AND VALUES options. The COLLECT ACTIVITY DATA WITHOUT DETAILS option is not sufficient, it will not capture the statement text which is required by the Design Advisor.

To import activity information from the activity event monitor tables into the Design Advisor, run the `db2advis` command with the `-wlm` parameter, followed by additional parameters:

1. The activities event monitor name
2. Optional: the workload or service class name
3. Optional: the start time and end time

For example, to import information about all the activities collected by the DB2ACTIVITIES event monitor in the SAMPLE database, use the following command:

```
db2advis -d SAMPLE -wlm DB2ACTIVITIES
```

**Note:** You can only import information from activities event monitor tables through the Design Advisor command line interface.

**Canceling activities**

If an activity is consuming too many resources, or is running too long, you can cancel it. Canceling an activity is gentler than forcing the application that submitted the activity. A canceled activity returns SQL4725N to the user, but does not end the connection or affect any other user activity. Forcing the application ends both the connection and user activities.

**About this task**

You can only explicitly cancel an activity if a coordinator activity is currently working on a request for the activity. If you cancel an activity in the IDLE state (that is, no requests are being processed), the activity is placed in the CANCEL_PENDING state and is canceled on the next request that is received. For example, if you attempt to cancel a CURSOR activity between fetches, the SQL4725N error is not returned to the user until the next fetch after the cancel.

All user activities are cancelable, including the load utility and stored procedures.

**Procedure**

1. Identify the activity that you want to cancel. You can use the `WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES` table function to identify the activities running in an application. You can also use the `MON_GET_ACTIVITYDETAILS_COMPLETE` table function to view additional
details about a particular activity if the information in
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES is not sufficient to
identify the work that the activities are performing.

2. Cancel the activity using the WLM_CANCEL_ACTIVITY stored procedure. The
stored procedure takes the following arguments: application_handle, uow_id, and
activity_id. For an example of how to use this stored procedure, see “Scenario:
Identifying activities that are taking too long to complete” on page 352.

Guidelines for capturing information about and investigating a rogue
activity

This topic provides guidelines for capturing information about, and investigating, a
rogue activity.

First establish a set of criteria for what you would consider a rogue activity. For
example:

- An activity in that runs in a service class for activities with a low estimated cost,
  and runs for more that 1 hour
- An activity that returns an unusually large number of rows
- An activity that consumes an unusually high amount of temporary table space

Then create thresholds that describe these criteria and contain a COLLECT
ACTIVITY DATA WITH DETAILS action. When the threshold is violated,
information about the activity that violated the threshold is sent to the active
ACTIVITIES event monitor when the activity completes.

For example, to collect information about any database activity that runs for more
than 3 hours, create a threshold such as the following threshold:

```
CREATE THRESHOLD LONGRUNNINGACTIVITIES
    FOR DATABASE ACTIVITIES ENFORCEMENT DATABASE
    WHEN ACTIVITYTOTALTIME > 3 HOURS COLLECT ACTIVITY DATA WITH DETAILS
    CONTINUE
```

Monitoring with DB2 workload manager is lightweight, if you are careful to apply
it only to a small subset of your queries, as shown in the example, where only
queries running for at least three hours are monitored. You can refine this example
further by creating a threshold not at the global database level, but at the level of a
user-defined superclass. If this more narrowly scoped monitoring suits your
purpose, it can further reduce the cost of monitoring and it will provide
information only at the level you need:

```
CREATE SERVICE CLASS LONGQUERIES
    PREFETCH PRIORITY LOW

CREATE THRESHOLD LONGRUNNINGACTIVITIES2
    FOR SERVICE CLASS LONGQUERIES ACTIVITIES ENFORCEMENT DATABASE
    WHEN ACTIVITYTOTALTIME > 3 HOURS COLLECT ACTIVITY DATA WITH DETAILS
    CONTINUE
```

After your data server has performed some work, you can analyze the information
that is written to the threshold violations and activities event monitors. DML
activities also have their statement text and compilation environment information
written to the activities event monitor, so you can run DB2 EXPLAIN on them to
further investigate the performance of the activity.
Workload management performance modelling

The workload on your system can be modelled as a set of activities that arrive at the system at a rate governed by an arrival rate distribution for activities (often measured as its inverse, the inter-arrival time distribution) and the amount of time activities spend executing in the system following a service time distribution.

Inter-arrival time is the time between the arrival of one activity and the arrival of the next activity. Service time is the time that an activity spends executing on the system. For example, if you submit a query at time 0 seconds, it spends 2 seconds in a queue, and it finishes at time 5 seconds, the service time is $5 - 2 = 3$ seconds. Service time assumes no other work executing on the system (that is, it is not the observed execution time, but rather the time it would take to execute the activity in isolation). The service time distribution can be approximated for DML activities using the estimated cost in timerons, which considers both processor and I/O time for an activity.

You can build a workload model for your system by measuring the inter-arrival time distribution and the service time distribution of the activities on the system. Inter-arrival time distributions and approximate service time distributions (using estimated cost) can be obtained by using extended aggregate activity statistics for service subclasses or work classes (using work actions) and a statistics event monitor. These statistics are not collected by default. For more information, see:

- “Statistics for DB2 workload management objects” on page 246
- A gentle introduction to histograms
- Understanding the six histograms of DB2 workload management
- Visualizing and deriving statistics from DB2 histograms using SQL

Example: Capturing information about an activity for later analysis

You can use workload management features to capture information about an activity for later analysis.

Assume that you have a stored procedure called MYSHEMA.MYSLOWSTP and that it is running more slowly than usual. You begin to receive complaints about this situation and decide to investigate the cause of the slowdown. If investigating while the stored procedure is running is impractical, you can capture information about the stored procedure activity and any activities nested in it.

Assuming that you have an active activities event monitor called DB2ACTIVITIES, you can create a work class for CALL statements that apply to the schema of the MYSHEMA.MYSLOWSTP stored procedure. Then you can create a work action to map the CALL activity and all nested activities to a service class that has activity collection enabled. The CALL activity, and any activities nested in it, are sent to the event monitor. Following are examples of the DDL required to create the DB2 workload management objects:

```
CREATE SERVICE CLASS SC1;
CREATE WORKLOAD WL1 APPLNAME ('DB2BP') SERVICE CLASS SC1;
CREATE SERVICE CLASS PROBLEMQUERIES UNDER SC1 COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS;
CREATE WORK CLASS SET PROBLEMQUERIES (WORK CLASS CALLSTATEMENTS WORK TYPE CALL ROUTINES IN SCHEMA MYSHEMA);
CREATE WORK ACTION SET DATABASEACTIONS FOR SERVICE CLASS SC1 USING WORK CLASS SET PROBLEMQUERIES (WORK ACTION CAPTURECALL ON WORK CLASS CALLSTATEMENTS MAP ACTIVITY WITH NESTED TO PROBLEMQUERIES);
```
After the MYSHEMA.MYSLOWSTP stored procedure runs, you can issue the following query to obtain the application handle, the unit of work identifier, and the activity identifier for the activity:

```
SELECT AGENT_ID,
UOW_ID,
ACTIVITY_ID
FROM ACTIVITY_DB2ACTIVITIES
WHERE SC_WORK_ACTION_SET_ID = (SELECT ACTIONSETID
FROM SYSCAT.WORKACTIONSETS
WHERE ACTIONSETNAME = 'DATABASEACTIONS')
AND SC_WORK_CLASS_ID = (SELECT WORKCLASSID
FROM SYSCAT.WORKCLASSES
WHERE WORKCLASSNAME = 'CALLSTATEMENTS'
AND WORKCLASSSETID =
(SELECT WORKCLASSSETID FROM SYSCAT.WORKACTIONSETS WHERE ACTIONSETNAME
= 'DATABASEACTIONS'));
```

Assuming that the captured activity has an application handle of 1, a unit of work identifier of 2, and an activity identifier of 3, the following results are generated:

<table>
<thead>
<tr>
<th>AGENT_ID</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Using this information, you can issue the following query against the ACTIVITY_DB2ACTIVITIES and the ACTIVITYSTMT_DB2ACTIVITIES tables to determine where the activity spent its time:

```
WITH RAH (LEVEL, APPL_ID, PARENT_UOW_ID, PARENT_ACTIVITY_ID, UOW_ID, ACTIVITY_ID, STMT_TEXT, TIME_CREATED, TIME_COMPLETED) AS
(SELECT 1, ROOT.APPL_ID, ROOT.PARENT_UOW_ID, ROOT.PARENT_ACTIVITY_ID, ROOT.UOW_ID, ROOT.ACTIVITY_ID, ROOTSTMT.STMT_TEXT, ROOT.TIME_CREATED, ROOT.TIME_COMPLETED
FROM ACTIVITY_DB2ACTIVITIES ROOT, ACTIVITYSTMT_DB2ACTIVITIES ROOTSTMT
WHERE ROOT.APPL_ID = ROOTSTMT.APPL_ID AND ROOT.AGENT_ID = 1
AND ROOT.UOW_ID = ROOTSTMT.UOW_ID AND ROOT.UOW_ID = 2
AND ROOT.ACTIVITY_ID = ROOTSTMT.ACTIVITY_ID AND ROOT.ACTIVITY_ID = 3
UNION ALL
SELECT PARENT.LEVEL +1, CHILD.APPL_ID, CHILD.PARENT_UOW_ID, CHILD.PARENT_ACTIVITY_ID, CHILD.UOW_ID, CHILD.ACTIVITY_ID, CHILDSTMT.STMT_TEXT, CHILD.TIME_CREATED, CHILD.TIME_COMPLETED
FROM RAH PARENT, ACTIVITY_DB2ACTIVITIES CHILD, ACTIVITYSTMT_DB2ACTIVITIES CHILDSTMT
WHERE PARENT.APPL_ID = CHILD.APPL_ID AND PARENT.PARENT_UOW_ID = CHILD.PARENT_UOW_ID AND CHILD.ACTIVITY_ID = CHILDSTMT.ACTIVITY_ID AND PARENT.LEVEL < 64)
SELECT UOW_ID, ACTIVITY_ID, SUBSTR(STMT_TEXT,1,40),
TIMESTAMPDIFF(2, CHAR(TIME_COMPLETED - TIME_CREATED)) AS LIFE_TIME
FROM RAH
ORDER BY UOW_ID, ACTIVITY_ID;
```

The results would resemble the following ones:

<table>
<thead>
<tr>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>STMT_TEXT</th>
<th>LIFE_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>CALL SLOWPROC</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>SELECT COUNT(*) FROM ORG</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>SELECT * FROM MYHUGETABLE</td>
<td>999</td>
</tr>
</tbody>
</table>
The results indicate that the stored procedure is spending most of its time querying the MYHUGETABLE table. Your next step is to investigate what changes to the MYHUGETABLE table might cause queries running against it to slow down.

When many stored procedures run simultaneously, greater overhead is incurred when performing the analysis. To solve this problem, you can create a workload and service class for running a stored procedure that is issued by a specific authorization identifier, a specific application, or both. You can then use the preceding method to analyze the behavior of the stored procedure.
Chapter 5. Integration with operating system workload managers

If available, use DB2 workload management in conjunction with an operating system workload manager, which provides you with additional capabilities.

The point of integration between DB2 workload management and operating system workload managers is the DB2 service class. You create a mapping between a DB2 service class and an operating system workload manager class when you define a DB2 service class by using the OUTBOUND CORRELATOR option of the CREATE SERVICE CLASS or the ALTER SERVICE CLASS statement.

If the outbound correlator is set, all threads in the DB2 service class are associated with the operating system workload manager using the outbound correlator when the next activity begins.

Integration of AIX Workload Manager with DB2 workload management

On the AIX operating system, the optional integration between DB2 service classes and AIX WLM classes permits you to control the amount of processor resource allocated to each service class.

Implementing AIX WLM controls may not be needed to meet your performance objectives, but even if you do not need to exercise AIX WLM, the operating system statistics provided by AIX WLM per AIX class are often useful for monitoring and tuning efforts.

AIX WLM assigns relative or absolute amounts of processor resource as shares to classes which benefit from controls that you can change dynamically and that become effective immediately. If relative AIX CPU shares do not provide the level of control you require, you also have the choice of assigning hard maximum percentage of CPU resource. By doing so, you surrender some of the flexibility of relative CPU allocation, which is useful during off-peak times, but you also gain excellent and guaranteed control with a hard maximum limit on CPU time resource allocation.

Recommended mappings between DB2 service classes and AIX classes

Use a 1:1 mapping of DB2 service classes to AIX Workload Manager service classes to take advantage of AIX WLM processor controls. By having a 1:1 mapping between DB2 service classes and AIX Workload Manager service classes, you can adjust the AIX processor resource for each DB2 service class individually to meet your business priority goals.

The following figure shows the integration of DB2 workload management with the AIX Workload Manager. Note the 1:1 mapping between each DB2 service class and AIX Workload Manager service class at the service superclass and service subclass levels.
When a DB2 environment consists of a single database in a single DB2 instance, such as the example portrayed in the previous figure, it is possible to map directly between DB2 service classes and AIX Workload Manager classes. Each DB2 service superclass can have a corresponding AIX Workload Manager service superclass and each DB2 service subclass can map to a corresponding AIX service subclass.

In situations where the DB2 environment consists of multiple databases and DB2 instances, several levels might be candidates for resource control. Because the AIX Workload Manager supports a two-level hierarchy, that is, superclass and subclass, only two levels of a DB2 environment can be mapped to AIX Workload Manager classes at any time. The following figure shows one way to achieve a 1:1 mapping with multiple databases, each with multiple superclasses. Here, each database has its own AIX Workload Manager superclass and each DB2 service superclass is mapped to an AIX Workload Manager subclass.

*Figure 52. Integration of DB2 workload management with the AIX Workload Manager*
An alternative configuration is to map each DB2 service superclass to its own AIX Workload Manager superclass, which results in four superclasses in this example. In this situation, the database level of resource control is represented explicitly in the AIX Workload Manager service class definitions.

The following figure shows one way to achieve the 1:1 mapping in the situation where you have multiple databases, each with service superclasses and service subclasses. Here, each database corresponds to an AIX superclass and each DB2 service subclass is mapped to an AIX Workload Manager subclass. The DB2 service superclass is not shown explicitly in the AIX Workload Manager service class definitions.

Figure 53. DB2 service classes mapped to AIX classes (with DB2 service superclasses only)
Figure 54. DB2 service classes mapped to AIX Workload Manager classes (with DB2 service subclasses)
Defining mappings between DB2 service classes and AIX classes

Mapping between DB2 service classes and AIX Workload Manager classes is specified for the DB2 service class using the OUTBOUND CORRELATOR keyword of the CREATE SERVICE CLASS or the ALTER SERVICE CLASS statements.

The steps for setting up the AIX Workload Manager classes with the DB2 data server are:
1. Create the DB2 service superclasses and service subclasses, and specify the OUTBOUND CORRELATOR tags.
2. Create the corresponding AIX classes.
3. Create the associated AIX Workload Manager rules files to contain the DB2 workload management to AIX Workload Manager mappings using the OUTBOUND CORRELATOR tags under the tag columns.
4. Start the AIX Workload Manager.
5. If required, set this AIX Workload Manager configuration as active.

When a thread joins a DB2 service class, the DB2 data server calls the appropriate AIX Workload Manager API to associate the thread to the corresponding AIX service class. The DB2 data server sends the thread’s target AIX service class to the AIX Workload Manager by passing it the application tag set in the OUTBOUND CORRELATOR parameter.

You must ensure that the AIX Workload Manager is properly installed, configured, and active. If the DB2 data server cannot communicate with the AIX Workload Manager, a message is logged to the db2diag log files and DB2 administrator log. The database activity continues.

The DB2 data server cannot detect whether the OUTBOUND CORRELATOR value that it passes to the AIX Workload Manager is recognized by the AIX Workload Manager. You must verify that the value specified for the DB2 service class matches the application tags that map DB2 threads to the AIX service classes. If the OUTBOUND CORRELATOR value is not recognized by the AIX Workload Manager, the database activity continues to execute.

Other points to note are:
- DB2 service classes cannot work with the AIX Workload Manager inheritance feature. Inheritance is the default setting for an AIX service class; inheritance must be explicitly disabled by setting the inheritance attribute to NO. AIX Workload Manager inheritance forces all child threads and processes to map to the same class as the parent thread or process. If inheritance is enabled, DB2 workload management cannot change the AIX Workload Manager class of a thread by using tagging. This restriction makes any integration of DB2 workload management and the AIX Workload Manager unusable. The DB2 data server cannot detect whether AIX Workload Manager inheritance is enabled and does not issue an error message if inheritance is enabled.
- DB2 service classes are not compatible with the AIX Workload Manager manual assignment feature. With the manual assignment feature, users can manually assign a process to a specific AIX Workload Manager class. By manually assigning the DB2 process, all threads in the process are assigned to a target AIX Workload Manager class, the DB2 service class mapping logic is defeated and results are not predictable.
For more information on the AIX Workload Manager, see the AIX Information Center at http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp

**Setting processor controls on AIX classes**

The AIX Workload Manager can be used to control the amount of processor resource allocated to each service class. Options include setting a minimum, maximum, or relative proportion share of processor resource for each service class.

When integrating the AIX Workload Manager with DB2 Workload Management, only processor resource allocation is supported. You should not set memory and I/O settings for the AIX classes. DB2 database-level memory is shared among all agents from different DB2 service classes, so you cannot divide memory allocation between different service classes. AIX-level I/O control does not support the DB2 engine threaded model. To control I/O, you can use the prefetcher priority attribute of a DB2 service class to differentiate I/O priorities between different DB2 service classes.

AIX Workload Manager settings should be consistent on all physical computers that participate in an instance. For example, if the resource setting for an AIX service class is set high on one computer, the same setting should be used for that AIX service class on all other computers. If the resource usage settings are inconsistent across computers, requests running in the same AIX service class will exhibit different performance levels on different database members. This situation can lead to poor overall throughput for connections in an AIX service class.

**Integration of Linux workload management with DB2 workload management**

On the Linux operating system, the optional integration between DB2 service classes and Linux classes (control groups) permits you to control the amount of processor resource allocated to each service class. If enabled, all threads running in a DB2 service class are mapped to a Linux class where they are subject to the processor resource controls you define.

To make use of Linux workload management support, you require a Linux kernel version 2.6.26 or later on a 64-bit system and the libcgroup library package.

Linux workload management supports a hierarchy of classes with superclasses and subclasses, with processor shares for subclasses divided in proportion to the shares of the parent class. These shares provide a method of control over processor resource such that all threads in the system will always run, but the amount of processor time each thread receives is dependent on the number of shares assigned to the Linux class.

Processor resource on the Linux operating system is assigned in shares relative to the Linux workload management default class, which by default has a processor share at a value of 1024. If you define no other Linux classes, all threads run in this default class. If you define a class that has a share value equal to 1024, then this class receives the same amount of processor resource as the Linux default class with the default processor share. Similarly, a class with a share of 2048 receives a target processor usage quota twice that of the default class. On more complex systems, you should consider raising the processor share of the Linux default class, which improves the granularity for shares across the system so that you can assign processor resources more accurately.
Recommended mappings between DB2 service classes and Linux classes

You should use a 1:1 mapping between DB2 service classes and Linux classes which permits you to adjust the Linux processor shares assigned to activities in each DB2 service class individually according to business priority. It is important that you associate every DB2 service class with a Linux WLM class, either by setting an outbound correlator for each service superclass and subclass, or through inheritance from the parent service class for subclasses. This includes the default SYSDEFAULTSYSTEMCLASS, SYSDEFAULTMAINTENANCECLASS and SYSDEFAULTUSERCLASS service classes.

The following figure shows how two DB2 service subclasses under the same user defined service superclass can get mapped 1:1 to Linux subclasses under a common superclass. In this example, the work identified and assigned by two workloads for each DB2 service subclass is subject to the processor resource controls imposed by the corresponding Linux subclasses (DB2_SUBCLASSA, DB2_SUBCLASSB). Also shown are three Linux classes that correspond to the default DB2 workload management service classes (DB2_DEF_USER, DB2_DEF_SYSTEM, DB2_DEF_MAINT). If you integrate DB2 workload management with Linux workload management, you should always create these additional Linux classes to match the default DB2 service classes. To avoid any bottleneck, the Linux class corresponding to the DB2 default system class should receive more processor shares than any other Linux class that DB2 activities map to, whilst the Linux class corresponding to the default maintenance class should receive less processor shares.

Figure 55. Integration of DB2 workload management with Linux workload management
Defining mappings between DB2 service classes and Linux workload management classes

The steps for integrating DB2 workload management with Linux workload management, which runs as an operating system service, are as follows:

1. Define the Linux classes, class permissions, and processor shares by editing the /etc/cgconfig.conf control groups configuration file. What Linux classes you create depends on the conditions dictated by your business priorities for the work your data server performs. If you want to apply processor resource based on the source of certain work, for example, create a Linux class to match the DB2 service class that work is going to be assigned to by the workload identifying the work. Define an entry for each Linux class corresponding to the DB2 service class to be created that you want to use for the mapping. The following sections must be provided in the /etc/cgconfig.conf configuration file:
   - group: The Linux class name. For example, if you specify group _class1, you create a superclass _class1. If you specify group _class1/_subclass1, you create the subclass _subclass1 under the superclass _class1.
   - perm: The permissions section that determines who can control what threads are assigned to a Linux class and who can change the processor shares of classes in the /etc/cgconfig.conf configuration file.
- task: The user ID (\texttt{uid}) and group ID (\texttt{gid}) whose threads can run in the Linux workload management class. To enable Linux workload management to work with DB2 workload management, you should set \texttt{uid} to the DB2 instance owner user ID.

- admin: The user ID (\texttt{uid}) and group ID (\texttt{gid}) that can change processor shares for a Linux workload management class.

- cpu: The processor shares definition section
  - cpu.shares: The share assigned to this Linux class relative to the default class

The \texttt{/etc/cgconfig.conf} configuration file must contain these sections in the following format:

\begin{verbatim}
# Superclass name
group _name
{
  perm
  {
    task
    {
      uid = db2Inst1;
      gid = db2Iadm1;
    }
    admin
    {
      uid = db2Inst1;
      gid = db2Iadm1;
    }
  }
  cpu
  {
    cpu.shares = 1024;
  }
}
\end{verbatim}

2. Start the Linux workload management service daemon with the \texttt{service cgconfig start} command, then start your DB2 data server with the \texttt{db2start} command.

3. To map a DB2 service class to one of the Linux classes, include the Linux class name in the OUTBOUND CORRELATOR clause when you create or alter the service class, which associates threads from the DB2 service class with the external Linux class.

4. If you want to find out what threads are assigned to a particular Linux class, you can use the \texttt{cat} command on the \texttt{/cgroup/class\_name/tasks} file, where \texttt{class\_name} represents the name of the Linux class you are interested in. All threads that are not mapped to a user-defined Linux class are assigned to the Linux default class, which you can find at \texttt{MOUNTPOINT/sysdefault}, where \texttt{MOUNTPOINT} is defined in the \texttt{cgconfig.conf} configuration file.

5. To add or remove Linux classes, you must stop the Linux workload management service with the \texttt{service cgconfig stop} command, make your changes, and then restart the service. Note that stopping the service affects the entire system, because all tasks are moved to the default class. If you used the \texttt{/etc/init.d/cgred} script to start the service daemon, issue \texttt{/etc/init.d/cgred stop} to stop it.

For the integration with DB2 workload management to work, you must ensure that the Linux workload management service is properly installed, configured, and active. If the DB2 data server cannot communicate with the Linux workload
management service, a message is logged to the db2diag log files and DB2 administrator log. Database activities will continue to execute.

The DB2 data server cannot detect whether the outbound correlator that it passes to external workload managers is recognized by Linux workload management. You must verify that the OUTBOUND CORRELATOR value specified for a DB2 service class matches the Linux class name so that DB2 threads are mapped to the Linux class. If an outbound correlator is not recognized, database activities will continue to execute.

**Example**

The following example illustrates how you can make use of Linux workload management processor controls by integrating with DB2 workload management. In this example, we create two user-defined DB2 service classes, one for batch applications (BATCHAPPS) and one for online applications (ONLINEAPPS). For simplicity, this example does not show the default service classes, which should be included in an implementation that creates the recommended 1:1 mapping between DB2 service classes and Linux classes. Because response time is critical for the online applications, we want the ONLINEAPPS service class to receive three times the amount of processor shares relative to work that runs in the Linux default class (3 x 1024 = 3072 shares). Batch applications have a lower business priority, and the BATCHAPPS class should be assigned half the processor resource of work that runs in the Linux default class (1024 / 2 = 512 shares). All other work on the system will run in the Linux default class. Note that this example does not create Linux classes corresponding to the three default DB2 workload management service classes.

To create this setup, first create the two corresponding Linux classes _BATCHAPPS and _ONLINEAPPS and set their relative processor shares by editing the /etc/cgconfig.conf tasks file. After editing, the tasks file contains the following two entries, one for each Linux class:

```bash
# Superclass ONLINEAPPS
group _ONLINEAPPS
{
  perm
  {
    task
    {
      uid = db2inst1;
      gid = db2iadm1;
    }
    admin
    {
      uid = db2inst1;
      gid = db2iadm1;
    }
  }

  cpu
  {
    # 3 x 1024 = 3072 shares
    cpu.shares = 3072;
  }
}

# Superclass BATCHAPPS

group _BATCHAPPS
{
  perm
```
The absolute processor time in percent assigned to each Linux class as processor shares is as follows:

<table>
<thead>
<tr>
<th>Linux class</th>
<th>Shares</th>
<th>Absolute processor time in percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default class</td>
<td>1024 (default)</td>
<td>1024 / 4608 = 22%</td>
</tr>
<tr>
<td>_ONLINEAPPS</td>
<td>1024 x 3 = 3072</td>
<td>3072 / 4608 = 67%</td>
</tr>
<tr>
<td>_BATCHAPPS</td>
<td>1024 x ½ = 512</td>
<td>512 / 4608 = 11%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1024 + 3072 + 512 = 4608 shares</td>
<td></td>
</tr>
</tbody>
</table>

Once the Linux WLM classes are created, you can start the Linux workload management service:
```
service cgconfig start
```

Next, create the associated DB2 service classes with the following statements:
```
DB2 CREATE SERVICE CLASS BATCHAPPS OUTBOUND CORRELATOR '__BATCHAPPS'
DB2 CREATE SERVICE CLASS ONLINEAPPS OUTBOUND CORRELATOR '__ONLINEAPPS'
```

To find out which threads are running in a Linux class, issue the cat command. For the business critical _ONLINEAPPS Linux class, the command and output look as follows. You can see that there are six thread running in this Linux class:
```
cat /cgroup/__ONLINEAPPS/tasks
```

1056
1087
1107
985
1036
1205
Chapter 6. Tutorial for DB2 workload management

The exercises in this tutorial were designed to provide you with a hands-on introduction to DB2 workload management. Each exercise highlights one or more of the workload management features available with DB2 workload management.

These exercises provide some guidance for using DB2 workload management features which you can adapt for your own purposes, but you should note that the initial configuration you chose for your own data server may differ and should be based on your specific workload management objectives.

Before you begin

This tutorial is designed to be run against the SAMPLE database and, unless noted otherwise, requires DBADM or WLMADM authority (or SQLADM authority if only the COLLECT ACTIVITY DATA clause is specified). You should also start the instance and activate the SAMPLE database before continuing:

```
db2start
db2 activate db sample
```

Some of the command and query statements shown in these exercises are quite long. You can find most of these statements in the text file `wlm-tutorial-steps.txt`, which you can copy from when working through the exercises. The scripts representing the workloads that are required for the different exercises are also included.

Both `wlm-tutorial-steps.txt` and the workload scripts can be found here.

Exercise 1: Getting started with basic monitoring using default DB2 workload management objects

This exercise demonstrates the basic types of monitoring information that can be obtained from the default workload and service class objects.

Estimated time: 20-25 minutes

By default, the user workload (SYSDEFAULTUSERWORKLOAD) and a default user service class (SYSDEFAULTUSERCLASS) are always created for each database. These default objects can be used to take advantage of the new DB2 workload management monitoring features without having to create any user defined workloads or service classes. If no user defined workloads and service classes are created, all user activities will be associated with these default objects.

There are two separate features of monitoring that are demonstrated by this exercise:

1. The ability to collect aggregate statistics for all activities that run in a service class. Aggregate activity statistics provide an inexpensive way of looking at work in a service class as a whole. They show information like the number of activities that ran in the service class, and the average lifetime of those activities.

2. The ability to capture information about individual activities. Activity information can be useful when investigating the performance or behavior of a
particular activity. Activity information includes things such as statement text, compilation environment, etc. Activity information is more expensive to collect than aggregate activity statistics and is usually targeted towards a specific subset of activities.

**Step 1: Create and enable event monitors**

Connect to the database and create and enable event monitors for activities and statistics.

```
CONNECT TO SAMPLE
CREATE EVENT MONITOR DB2ACTIVITIES FOR ACTIVITIES WRITE TO TABLE
CREATE EVENT MONITOR DB2STATISTICS FOR STATISTICS WRITE TO TABLE
SET EVENT MONITOR DB2ACTIVITIES STATE 1
SET EVENT MONITOR DB2STATISTICS STATE 1
```

**Step 2: Collect individual activities**

Enable collection of individual activities using the COLLECT ACTIVITY DATA clause on the CREATE or ALTER WORKLOAD STATEMENT. When the COLLECT ACTIVITY DATA clause is specified for a workload, information about any activity submitted by an occurrence of that workload will be sent to the active ACTIVITIES event monitor when the activity completes. The COLLECT ACTIVITY DATA clause permits you to specify how much information should be collected by applying one of the following options:

- **WITHOUT DETAILS**: Collect activity information without statement and compilation environment.
- **WITH DETAILS**: Collect activity information including statement and compilation environment.
- **WITH DETAILS AND VALUES**: Collect activity information including statement and compilation environment, and input data values.

For this exercise, you will specify the WITH DETAILS clause so that the statement text information is captured.

```
ALTER WORKLOAD SYSDEFAULTUSERWORKLOAD
  COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
```

In this example activity data is collected for the default user workload. This results in information about all user activities being collected since no other user defined workloads are currently active. This would be too expensive in a production environment. A better approach would be to isolate the activities of interest using a specific user defined workload or service class and apply the COLLECT ACTIVITY DATA clause to that workload or service class only.

Additional Information: The COLLECT ACTIVITY DATA clause can also be specified on a service class, work class (using a work action) or a threshold. If the clause is specified for a service class, information will be collected for any activity that runs in the service class. If it is specified for a work class (using a work action), any activity that has the work action applied to it will be collected. If the clause is specified for a threshold, activity information will be collected if the threshold is violated.
Step 3: Collect aggregate activity statistics

Enable collection of aggregate activity statistics for the default subclass under the default user service class using the COLLECT AGGREGATE ACTIVITY DATA clause. When this clause is specified, aggregate statistics will be maintained in memory for the corresponding service class (for example, statistics such as average activity lifetime). The statistics can be viewed using the service subclass statistics table function, or can be collected and sent to the active statistics event monitor for later analysis.

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

Additional Information: There is a set of statistics collected by default for all DB2 workload management objects. The COLLECT AGGREGATE ACTIVITY DATA clause enables collection of a number of additional optional statistics, such as the activity lifetime histogram.

In this example all user activities will be run in the SYSDEFAULTSUBCLASS service subclass under the SYSDEFAULTUSERCLASS service super class because no user defined service classes have been created. Therefore, information will be collected for all user activities.

Step 4: Run some activities

Run some activities, which will result in statistics being updated and the activities being collected.
```
db2 -o -tvf work1.db2
db2 -o -tvf work2.db2
```

The scripts representing applications (such as work1.db2 and work2.db2) disconnect you from the database, so that after running them you will need to reconnect.

Step 5: View statistics

You can view the service class statistics using the WLM_GET_SERVICE_SUBCLASS_STATS table function. For example:
```
CONNECT TO SAMPLE

SELECT VARCHAR(SERVICE_SUPERCLASS_NAME, 30) AS SUPERCLASS,
    VARCHAR(SERVICE_SUBCLASS_NAME, 30) AS SUBCLASS,
    LAST_RESET,
    COORD_ACT_COMPLETED_TOTAL,
    COORD_ACT_REJECTED_TOTAL,
    COORD_ACT_ABORTED_TOTAL,
    COORD_ACT_LIFETIME_AVG
FROM TABLE(SYSPROC.WLM_GET_SERVICE_SUBCLASS_STATS('SYSDEFAULTUSERCLASS',
    'SYSDEFAULTSUBCLASS', -1)) AS T
```

The output from this query will look something such as the following:
```
SUPERCLASS SUBCLASS LAST_RESET
COORD_ACT_COMPLETED_TOTAL COORD_ACT_REJECTED_TOTAL COORD_ACT_ABORTED_TOTAL
COORD_ACT_LIFETIME_AVG
----------------- ------------------ -------------- ------------------ ------------------ ----------------------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2007-07-18-16.03.51.752190
74 0 0 +1.40288000000000E+002
```

1 record(s) selected.
The COORD_ACT_COMPLETED_TOTAL column indicates how many activities have completed successfully in this service class. The last reset time indicates the last time that statistics were reset for this service class.

Additional Information: If you do not enable aggregate activity statistics for a service class using the COLLECT AGGREGATE ACTIVITY DATA clause, some statistics reported by the WLM_GET_SERVICE_SUBCLASS_STATS table function will be NULL.

**Step 6: Send statistics to event monitor**

Use the WLM_COLLECT_STATS stored procedure to send the statistics for all DB2 workload management objects to the active statistics event monitor. When statistics are collected and sent to the statistics event monitor, the values are reset.

```sql
CALL SYSPROC.WLM_COLLECT_STATS()
```

Additional Information: If there is no active statistics event monitor, you can still use the WLM_COLLECT_STATS procedure to reset the statistics, but the current values will be lost. It is possible to automate workload management statistics collection using the WLM_COLLECT_INT database configuration parameter. If you set this parameter to a nonzero value, workload management statistics will be collected automatically every `wlm_collect_int` minutes (as if you manually invoked the WLM_COLLECT_STATS procedure every `wlm_collect_int` minutes).

**Step 7: View statistics again**

Invoke the WLM_GET_SERVICE_SUBCLASS_STATS table function again. Note that the LAST_RESET timestamp has been updated and the statistics have been reset.

```sql
SELECT VARCHAR(SERVICE_SUPERCLASS_NAME, 30) AS SUPERCLASS,
       VARCHAR(SERVICE_SUBCLASS_NAME, 30) AS SUBCLASS,
       LAST_RESET,
       COORD_ACT_COMPLETED_TOTAL,
       COORD_ACT_REJECTED_TOTAL,
       COORD_ACT_ABORTED_TOTAL,
       COORD_ACT_LIFETIME_AVG
FROM TABLE(SYSPROC.WLM_GET_SERVICE_SUBCLASS_STATS('SYSDEFAULTUSERCLASS',
                                                    'SYSDEFAULTSUBCLASS', -1)) AS T
```

The output will look something like:

```
SUPERCLASS  SUBCLASS  LAST_RESET  COORD_ACT_COMPLETED_TOTAL  COORD_ACT_REJECTED_TOTAL  COORD_ACT_ABORTED_TOTAL  COORD_ACT_LIFETIME_AVG
-------------------------------------------------------------------------------------------------------------------------------------
SYSDEFAULTUSERCLASS  SYSDEFAULTSUBCLASS  2007-07-18-16.04.03.505818  0  0  +0.00000000000000E+000
```

1 record(s) selected.

**Step 8: View service class statistics collected by the statistics event monitor**

The WLM_COLLECT_STATS procedure sent the service class statistics to the statistics event monitor. You can look at the statistics that were collected by the event monitor using statement such as the following:
SELECT VARCHAR(SERVICE_SUPERCLASS_NAME, 30) AS SUPERCLASS,
       VARCHAR(SERVICE_SUBCLASS_NAME, 30) AS SUBCLASS,
       LAST_WLM_RESET,
       STATISTICS_TIMESTAMP,
       COORD_ACT_COMPLETED_TOTAL,
       COORD_ACT_REJECTED_TOTAL,
       COORD_ACT_ABORTED_TOTAL,
       COORD_ACT_LIFETIME_AVG
FROM SCSTATS_DB2STATISTICS

The output will look something like:

<table>
<thead>
<tr>
<th>SUPERCLASS</th>
<th>SUBCLASS</th>
<th>LAST_WLM_RESET</th>
<th>STATISTICS_TIMESTAMP</th>
<th>COORD_ACT_COMPLETED_TOTAL</th>
<th>COORD_ACT_REJECTED_TOTAL</th>
<th>COORD_ACT_ABORTED_TOTAL</th>
<th>COORD_ACT_LIFETIME_AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTSYSTEMCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2007-07-18-16.03.46.33724</td>
<td>2007-07-18-16.04.03.505818</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>SYSDEFAULTMAINTENANCECLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2007-07-18-16.03.46.334301</td>
<td>2007-07-18-16.04.03.505818</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2007-07-18-16.03.51.752190</td>
<td>2007-07-18-16.04.03.505818</td>
<td>75</td>
<td>0</td>
<td>136</td>
<td></td>
</tr>
</tbody>
</table>

3 record(s) selected.

Every time statistics are sent to the event monitor, a statistics record will be created for each DB2 workload management object. Note the two timestamps LAST_WLM_RESET and STATISTICS_TIMESTAMP. The interval of time from LAST_WLM_RESET to STATISTICS_TIMESTAMP indicates the period of time over which the statistics in that record were collected. The STATISTICS_TIMESTAMP indicates when the statistics were collected. Note that the average activity lifetime statistic is only maintained for a service class if aggregate activity statistics are enabled using the COLLECT AGGREGATE ACTIVITY DATA clause.

**Step 9: View activity information**

Information about every individual activity associated with the default user workload was also collected by the activities event monitor, due to the specification of the COLLECT ACTIVITY DATA clause on the default workload in step 2. You can look at this activity information using a query such as the following:

```
SELECT VARCHAR(A.APPL_NAME, 15) as APPL_NAME,
       VARCHAR(A.TPMON_CLIENT_APP, 20) AS CLIENT_APP_NAME,
       VARCHAR(A.APPL_ID, 30) as APPL_ID,
       A.ACTIVITY_ID,
       A.UOW_ID,
       VARCHAR(S.STMT_TEXT, 300) AS STMT_TEXT
FROM ACTIVITY_DB2ACTIVITIES AS A,
     ACTIVITY_STMT_DB2ACTIVITIES AS S
WHERE A.APPL_ID = S.APPL_ID AND
     A.ACTIVITY_ID = S.ACTIVITY_ID AND
     A.UOW_ID = S.UOW_ID
```

The output will look something like:
Exercise 2: Isolating activities using service classes and workloads

This exercise demonstrates how to create service classes and how to send activities to a service class using a workload. It also demonstrates how to use some of the WLM monitoring features to determine the workload that activities are being mapped to and to get information about activities being run in a service class and under a workload.

Estimated time: 20-25 minutes

Service classes are the primary point of resource control for database activities. They are also useful for monitoring. For example, you can collect statistics for
activities in a particular service class to determine whether the performance goals for that service class are being met. By default, three default service classes (SYSDEFAULSYSTEMCLASS, SYSDEFAULTMaintenanceCLASS, and SYSDEFAULTUSERCLASS) are created for each database. If no user defined service classes are created, user activities are run under the default user service class (SYSDEFAULTUSERCLASS).

A workload is an entity that groups one or more units of work based on criteria such as system user ID, session user ID, etc. Workloads provide a means of assigning work to a service class so that the work can later be managed. A default user workload (SYSDEFAULUSERWORKLOAD) and a default administration workload (SYSDEFAULTADMWORKLOAD) are created for each database. If no user defined workloads are created, all user activities are associated with the default user workload.

There are four separate features that are demonstrated in this exercise:

- How to create a service class.
- How to create a workload.
- How to examine basic workload statistics.
- How to collect activity information for activities run under an individual workload.

**Step 1: Examine where activities are run with no user-defined service classes and workloads**

First examine where activities are executed if there is no user defined service class or workload. All DB2 activities are assigned to a workload and run in a service class. If no user defined service classes are created, activities run in the default subclass (SYSDEFAULTSUBCLASS) under the default user service class (SYSDEFAULTUSERCLASS) and if no user defined workloads are created, activities run under the default user workload (SYSDEFAULUSERWORKLOAD).

Run the work1.db2 and work2.db2 scripts and then examine the statistics for the SYSDEFAULTSUBCLASS of SYSDEFAULTUSERCLASS using the WLM_GETgetService_SUBCLASS_STATS.

```
    db2 -o -tvf work1.db2
    db2 -o -tvf work2.db2
    CONNECT TO SAMPLE
    SELECT VARCHAR(SERVICE_SUPERCLASS_NAME, 30) SUPERCLASS,
        VARCHAR(SERVICE_SUBCLASS_NAME, 30) SUBCLASS,
        COORD_ACT_COMPLETED_TOTAL
    FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('','',-1)) AS T
```

You will see output such as the following:

```
SUPERCLASS   SUBCLASS      COORD_ACT_COMPLETE
-------------- -------------- ------------------
SYSDEFAULSYSTEMCLASS 0        SYSDEFAULTSUBCLASS
SYSDEFAULTMaintenanceCLASS 0   SYSDEFAULTSUBCLASS
SYSDEFAULTUSERCLASS 75        SYSDEFAULTSUBCLASS
```

3 record(s) selected.
Note all the activities are run in the SYSDEFAULTUSERCLASS service super class.

Additional Information: There are 2 other service classes as well, SYSDEFAULTSYSTEMCLASS and SYSDEFAULTMAINTENANCECLASS. These service classes are used for internal maintenance and system level tasks. User activities will not run in these service classes. You may notice nonzero activity counts in these service classes as well if the DB2 data server has issued any internal activities.

Use the WLM_GET_WORKLOAD_STATS table function to view workload statistics to determine which workload the applications are being associated with.

```
SELECT SUBSTR(WORKLOAD_NAME, 1, 22) AS WL_DEF_NAME,
       WLO_COMPLETED_TOTAL,
       CONCURRENT_WLO_ACT_TOP
FROM TABLE(WLM_GET_WORKLOAD_STATS(CAST(NULL AS VARCHAR(128)), -2))
AS WLSTATS
```

The output will look something such as the following:

<table>
<thead>
<tr>
<th>WL_DEF_NAME</th>
<th>WLO_COMPLETED_TOTAL</th>
<th>CONCURRENT_WLO_ACT_TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>SYSDEFAULTADMWORKLOAD</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

2 record(s) selected.

Note there is one workload occurrence completed for both of the scripts (work1.db2 and work2.db2) as well as a workload occurrence for the connection used to execute the previous command.

**Step 2: Create a service class and workload**

Create a service class and then create a workload such that all activities run from the work1.db2 script get mapped to the newly created service class. When CLP executes a script, the CURRENT_CLIENT_APPLNAME special register value is set to "CLP script name".

```
CREATE SERVICE CLASS work1_sc
CREATE WORKLOAD work1_wl CURRENT CLIENT_APPLNAME('CLP work1.db2')
                   SERVICE CLASS work1_sc
```

Additional Information: There are a number of attributes that can be specified when creating a workload or a service class. For example, when creating a workload, you can identify the connection based on application name, session user, etc. For more information, refer the CREATE WORKLOAD and the CREATE SERVICE CLASS documentation.

**Step 3: Grant usage on workload**

Grant usage on the workload (requires ACCESSCTRL or SECADM authority).

```
GRANT USAGE ON WORKLOAD work1_wl TO PUBLIC
```

Additional information: A connection can be associated with a workload only if the session user has USAGE privilege on the workload. This is necessary to prevent users from changing connection attributes of their application in an attempt to run their work in a higher priority service class. Some connection attributes can be changed programmatically (using the sqleseti API, for example). In this exercise, we just grant USAGE privilege to PUBLIC. You would want to be more
discriminating on a real system. Since the sample will be run as DBADM, this step could be skipped altogether.

**Step 4: Reset statistics**

Reset the statistics using the WLM_COLLECT_STATS function, to clear the statistics collected.

```sql
CALL SYSPROC.WLM_COLLECT_STATS()
```

**Step 5: Run some activities**

Run both the `work1.db2` and the `work2.db2` scripts.

```sql
db2 -o -tvf work1.db2
```

```sql
db2 -o -tvf work2.db2
```

**Step 6: View workload and service class statistics**

Use the WLM_GET_WORKLOAD_STATS table function to view workload statistics to determine which workload the applications are being associated with.

```sql
CONNECT TO SAMPLE
SELECT SUBSTR(WORKLOAD_NAME, 1, 22) AS WL_DEF_NAME,
   WLO_COMPLETED_TOTAL,
   CONCURRENT_WLO_ACT_TOP
FROM TABLE(WLM_GET_WORKLOAD_STATS(CAST(NULL AS VARCHAR(128)), -2))
   AS WLSTATS
```

The output will look something such as the following:

<table>
<thead>
<tr>
<th>WL_DEF_NAME</th>
<th>WLO_COMPLETED_TOTAL</th>
<th>CONCURRENT_WLO_ACT_TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK1_WL</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>SYSDEFAULTADMWORKLOAD</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that one workload occurrence completed under WORK1_WL which is the `work1.db2` script. One workload occurrence completed under SYSDEFAULTUSERWORKLOAD which is the `work2.db2` script.

You may see a 2nd workload occurrence completed for the SYSDEFAULTUSER WORKLOAD which is the connection that was used to call the WLM_COLLECT_STATS procedure. WLM_COLLECT_STATS is an asynchronous procedure which might be completed before the statistics are actually collected and therefore might be included.

You can also use the WLM_GET_SERVICE_SUBCLASS_STATS table function to show which service class the activities are being run under as a result of creating the new workload.

```sql
SELECT VARCHAR(SERVICE_SUPERCLASS_NAME, 30) AS SUPERCLASS,
   VARCHAR(SERVICE_SUBCLASS_NAME, 23) AS SUBCLASS,
   COORD_ACT_COMPLETED_TOTAL, COORDACTCOMP
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS(='', '', -1)) AS T
```

The results looking something such as the following:

<table>
<thead>
<tr>
<th>SUPERCLASS</th>
<th>SUBCLASS</th>
<th>COORDACTCOMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTSYSTEMCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
</tr>
</tbody>
</table>
SYSDEFAULTMAINTENANCECLASS  SYSDEFAULTSUBCLASS  0
SYSDEFAULTUSERCLASS  SYSDEFAULTSUBCLASS  37
WORK1_SC  SYSDEFAULTSUBCLASS  37

Note the activities that completed under the WORK1_SC due to the WORK1_WL workload mapping.

**Step 7: Create another service class and workload**

Create a second service class and then create a workload such that all activities run from the work2.db2 application get mapped to the newly created service class. In addition, set up the workload so that it will collect some activity data. For this example, we just collect activity data without any additional details or values.

```
CREATE SERVICE CLASS work2_sc
CREATE WORKLOAD work2_wl
  CURRENT CLIENT_APPLNAME('CLP work2.db2')
  SERVICE CLASS work2_sc
  COLLECT ACTIVITY DATA
```

Additional information: When the COLLECT ACTIVITY DATA clause is specified for a workload, information about any activity submitted by an occurrence of that workload will be sent to the active ACTIVITIES event monitor when the activity completes. The COLLECT ACTIVITY DATA clause permits you to specify how much information should be collected by applying one of the following options:

- **WITHOUT DETAILS:** Collect activity information without statement and compilation environment (the default)
- **WITH DETAILS:** Collect activity information including statement and compilation environment.
- **WITH DETAILS AND VALUES:** Collect activity information including statement and compilation environment, and input data values.

**Step 8: Enable the activities event monitor**

Enable the event monitors for activities.

The activity event monitor was created in Exercise 1.

```
SET EVENT MONITOR DB2ACTIVITIES STATE 1
```

**Step 9: Reset statistics and run some activities**

Use the WLM_COLLECT_STATS stored procedure to reset the statistics again and run the work1.db2 and work2.db2 scripts again.

```
CALL SYSPROC.WLM_COLLECT_STATS()
db2 -o -tvf work1.db2
db2 -o -tvf work2.db2
```

**Step 10: View workload and service class statistics**

Use the WLM_GET_WORKLOAD_STATS table function again to determine which workload the applications are being associated with.

```
CONNECT TO SAMPLE
SELECT SUBSTR(WORKLOAD_NAME, 1, 22) AS WL_DEF_NAME,
```
The output will look something such as the following:

<table>
<thead>
<tr>
<th>WL_DEF_NAME</th>
<th>WLO_COMPLETED_TOTAL</th>
<th>CONCURRENT_WLO_ACT_TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK1_WL</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>WORK2_WL</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>SYSDEFAULTUSERWORKLOAD</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SYSDEFAULTADMWORKLOAD</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note this time both workload definitions have a workload occurrence run, once for each script.

You may or may not see a workload occurrence completed for the SYSDEFAULTUSERWORKLOAD depending on whether workload occurrence over which the call to the WLM_COLLECT_STATS procedure was submitted is closed before the statistics are collected.

Use WLM_GET_SERVICE_SUBCLASS_STATS again to show which service class the activities are being run under as a result of creating the new workload.

```sql
SELECT VARCHAR( SERVICE_SUPERCLASS_NAME, 30) SUPERCLASS,
       VARCHAR( SERVICE_SUBCLASS_NAME, 23) SUBCLASS,
       COORD_ACT_COMPLETED_TOTAL COORDACTCOMP
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('','',-1)) AS T
```

With the results looking something like:

<table>
<thead>
<tr>
<th>SUPERCLASS</th>
<th>SUBCLASS</th>
<th>COORDACTCOMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTSYSTEMCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
</tr>
<tr>
<td>SYSDEFAULTMAINTENANCECLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>1</td>
</tr>
<tr>
<td>WORK1_SC</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>37</td>
</tr>
<tr>
<td>WORK2_SC</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>37</td>
</tr>
</tbody>
</table>

Note this time service super class work2_sc has some activities run under it due to the WORK2_WL mapping. The one activity under SYSDEFAULTUSERCLASS is the query previously run on WLM_GET_WORKLOAD_STATS.

**Step 11: View the activity data collected**

Query the activity table for information on the activities that have been run. Note that only the activities from the work2.db2 script have been collected because only the work2_wl workload definition has the COLLECT ACTIVITY DATA attribute specified.

```sql
SELECT SUBSTR(WORKLOADNAME, 1, 20) WL_DEF_NAME,
       SUBSTR(APPL_NAME, 1, 20) APPL_NAME,
       SUBSTR(ACTIVITY_TYPE, 1, 10) ACT_TYPE
FROM SYSCAT.WORKLOADS, ACTIVITY_DB2ACTIVITIES
WHERE WORKLOADID = WORKLOAD_ID
```

The results look something like:

<table>
<thead>
<tr>
<th>WL_DEF_NAME</th>
<th>APPL_NAME</th>
<th>ACT_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK2_WL</td>
<td>db2bp</td>
<td>READ_DML</td>
</tr>
<tr>
<td>WORK2_WL</td>
<td>db2bp</td>
<td>READ_DML</td>
</tr>
<tr>
<td>WORK2_WL</td>
<td>db2bp</td>
<td>READ_DML</td>
</tr>
<tr>
<td>WORK2_WL</td>
<td>db2bp</td>
<td>READ_DML</td>
</tr>
<tr>
<td>WORK2_WL</td>
<td>db2bp</td>
<td>READ_DML</td>
</tr>
</tbody>
</table>
Step 12: Assign resources to service classes

Now that you have isolated the activities that are generated by these two scripts into separate service classes, you can assign resources to the service classes or monitor the activities that run in those service classes. If the work performed by the script `work2.db2` is more important than the work performed by the script `work1.db2`, you could use the ALTER SERVICE CLASS statement to decrease the amount of CPU that is given to work running in the WORK1_SC service class by setting the CPU limit for WORK1_SC, as shown in the following example. You must first enable the DB2 workload management dispatcher by setting the value of the `wlm_dispatcher` database manager configuration parameter to YES. The workload management dispatcher manages CPU resources that are allocated to DB2 user and maintenance service classes.

```
UPDATE DBM CFG USING WLM_DISPATCHER YES

ALTER SERVICE CLASS WORK1_SC CPU LIMIT 20
```

If you want to capture details about every individual activity that executes in the WORK2_SC service class, use the ALTER SERVICE CLASS statement to enable activity collection for that service class, as shown in the following example:

```
ALTER SERVICE CLASS SYSDEFAULTSUBCLASS UNDER WORK2_SC
    COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
```
**Step 13: Reset for next exercise**

Update workload work2_wl so that no activity data is collected, disable the event monitor and clean up the event monitor table, and call WLM_COLLECT_STATS() to reset the statistics.

```sql
ALTER WORKLOAD work2_wl
   COLLECT ACTIVITY DATA NONE
SET EVENT MONITOR DB2ACTIVITIES STATE 0
DELETE from ACTIVITY_DB2ACTIVITIES
CALL WLM_COLLECT_STATS()
```

---

**Exercise 3: Using thresholds to control rogue activities and using the threshold violation monitor**

This exercise demonstrates how you can use thresholds to detect resource misuse or the beginning of system overload by establishing limits over the consumption of a specific resource.

Estimated time: 15-20 minutes

If a threshold is violated, a specified action can be triggered. The supported actions are:

- **STOP EXECUTION**: Stop processing the activity that caused the threshold to be violated.
- **CONTINUE**: Continue processing
- Collect information about the activity that violated the threshold. This action can be specified in conjunction with the CONTINUE or STOP EXECUTION action.

Regardless of whether an activity that violates a threshold is stopped or permitted to continue running, a record of the violation is written to an active THRESHOLD VIOLATIONS event monitor (assuming one is defined in advance) each time a threshold is violated. The record contains information such as which threshold was violated, the time of the violation, and the threshold action.

This exercise demonstrates how thresholds can be used to detect or prevent rogue activities from running on your system and using up system resources. A rogue activity is any activity that uses an unexpectedly high amount of resources. For example, a query that runs for an abnormally long time, or returns an unexpectedly large result set.

**Step 1: Create a threshold violation event monitor**

Create and enable a write-to-table event monitor that will be used to capture the threshold violation information and enable the activity event monitor that was created in Exercise 1.

```sql
CREATE EVENT MONITOR threvio FOR THRESHOLD VIOLATIONS WRITE TO TABLE
   THRESHOLDVIOLATIONS(IN userspace1),
   CONTROL(IN userspace1)

SET EVENT MONITOR threvio STATE 1
SET EVENT MONITOR db2activities STATE 1
```
**Step 2: Create a workload**

Create a workload such that all activities run from the workth.db2 script will get mapped to the work1_sc service class.

The work1_sc service class already exists since it was created in Exercise 2.

```sql
CREATE WORKLOAD workth_wl
    CURRENT CLIENT_APPLNAME('CLP workth.db2')
    SERVICE CLASS work1_sc
```

**Step 3: Create thresholds**

Create two thresholds, one of which (th_estcost) is an ESTIMATEDSQLCOST threshold and another (th_sqlrows) is a SQLROWSRETURNED threshold and apply them to the service class you wish to control the activities for (in this case, work1_sc service class).

The th_estcost threshold specifies an upper bound (10000 timerons) for the optimizer-estimated cost (in timerons) for an activity running in the work1_sc service class. If any query with an estimated cost greater than 10000 timerons, tries to execute in the work1_sc service class, this threshold is violated and the query is not permitted to run.

The th_sqlrows threshold specifies that any activity running in the work1_sc service class can return at most 30 rows from the data server. If any query tries to return more than 30 rows, this threshold is violated, only 30 rows will be returned to the client and the query will be stopped. In addition, data about the activity that caused the threshold violation will be collected.

In either case, when an activity violates the threshold, a threshold violation record is written to the THRESHOLD VIOLATIONS event monitor as defined in step 1 and the execution of the activity is stopped (because of the STOP EXECUTION action). The application that submitted the activity will receive an SQL4712N error.

```sql
CREATE THRESHOLD th_estcost
    FOR SERVICE CLASS work1_sc ACTIVITIES
    ENFORCEMENT DATABASE
    WHEN ESTIMATEDSQLCOST > 10000
    STOP EXECUTION

CREATE THRESHOLD th_sqlrows
    FOR SERVICE CLASS work1_sc ACTIVITIES
    ENFORCEMENT DATABASE
    WHEN SQLROWSRETURNED > 30
    COLLECT ACTIVITY DATA WITH DETAILS AND VALUES
    STOP EXECUTION
```

Additional information: A threshold can be either predictive or reactive:

- **Predictive threshold:** The boundaries of a predictive threshold are checked before the activity starts running. To check whether a predictive threshold would be violated, the data server obtains usage estimates from the query compiler. For this example, the th_estcost threshold is a predictive threshold.

- **Reactive threshold:** The boundaries of a reactive threshold are checked while an activity is executing. Approximate runtime usage estimates of the controlled resource are used to evaluate the boundaries of reactive thresholds. The runtime usage estimates are not obtained continuously but rather at selected predefined checkpoints during the lifetime of the tracked work. For this example, the th_sqlrows is a reactive threshold.
Step 4: Run some activities

Run some activities, some of which violate the threshold upper bounds defined in the previous step.

db2 -o -tvf workth.db2

Note that the statements which violate the thresholds defined previously, fail with an error of SQL4712N/SQLSTATE 5U026.

Step 5: View the threshold violation event monitor

Information about every threshold violation is collected by the THRESHOLD VIOLATIONS event monitor. You can query the threshold violation information by issuing regular SQL statements against the threshold violation monitor table as shown in the following example.

CONNECT TO SAMPLE
SELECT APPL_ID,
        UOW_ID,
        ACTIVITY_ID,
        COORD_PARTITION_NUM AS COORDPART,
        THRESHOLD_PREDICATE,
        THRESHOLD_ACTION,
        TIME_OF_VIOLATION
FROM THRESHOLDVIOLATIONS_THREVIO
ORDER BY THRESHOLD_ACTION, THRESHOLD_PREDICATE, TIME_OF_VIOLATION

The output will look something such as the following:

<table>
<thead>
<tr>
<th>APPL_ID</th>
<th>UOW_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCAL.DB2.070821150008</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EstimatedSQLCost</td>
<td>Stop 2007-08-21-</td>
</tr>
<tr>
<td>11.00.11.000000</td>
<td></td>
</tr>
</tbody>
</table>

2 record(s) selected.

Step 6: View information for the activity that violated the threshold

Activity information is collected for any activity that violates a threshold that is defined with a COLLECT clause. Show the detailed information about the activities that violated a threshold using the following query:

SELECT VARCHAR(A.APPL_NAME, 15) as APPL_NAME,
        VARCHAR(A.TPMON_CLIENT_APP, 20) AS CLIENT_APP_NAME,
        A.ACTIVITY_ID,
        A.ACTIVITY_TYPE,
        A.WORKLOAD_ID,
        T.THRESHOLD_PREDICATE,
        A.QUERY_CARD_ESTIMATE,
        T.THRESHOLD_MAXVALUE,
        T.TIME_OF_VIOLATION,
        VARCHAR(A5. STMT_TEXT, 100) AS STMT_TEXT
FROM A
JOIN T ON A.ACTIVITY_ID = T.ACTIVITY_ID
WHERE T.THRESHOLD_ACTION = 'Stop';

Chapter 6. Tutorial for DB2 workload management

FROM THRESHOLDVIOLATIONS_THREVIO AS T,
ACTIVITY_DB2ACTIVITIES AS A,
ACTIVITY_STMT_DB2ACTIVITIES AS AS
WHERE T.APPL_ID = A.APPL_ID AND
T.UOW_ID = A.UOW_ID AND
T.ACTIVITY_ID = A.ACTIVITY_ID AND
A.APPL_ID = AS.APPL_ID AND
A.ACTIVITY_ID = AS.ACTIVITY_ID AND
A.UOW_ID = AS.UOW_ID

The output will look something such as the following:

<table>
<thead>
<tr>
<th>APPL_NAME</th>
<th>CLIENT_APP_NAME</th>
<th>ACTIVITY_ID</th>
<th>ACTIVITY_TYPE</th>
<th>WORKLOAD_ID</th>
<th>THRESHOLD_PREDICATE</th>
<th>QUERY_CARD_ESTIMATE</th>
<th>THRESHOLD_MAXVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2bp</td>
<td>CLP</td>
<td>3</td>
<td>READ_DML</td>
<td>3</td>
<td>SQLRowsReturned</td>
<td>41</td>
<td>3</td>
</tr>
</tbody>
</table>

0 2007-08-31-09.01.16.000000 SELECT * FROM SALES

Note that the activity that violated the th_estcost (EstimatedSqlCost) threshold is not shown. The reason is that the threshold did not specify the COLLECT ACTIVITY DATA clause, so that no activity data was collected for that activity.

**Step 7: Reset for next exercise**

Disable the event monitors that were enabled. Also disable and drop the th_estcost and th_sqlrows thresholds that were created.

SET EVENT MONITOR threvio STATE 0
SET EVENT MONITOR db2activities STATE 0

ALTER THRESHOLD th_estcost DISABLE
DROP THRESHOLD th_estcost

ALTER THRESHOLD th_sqlrows DISABLE
DROP THRESHOLD th_sqlrows

Also clean up the activities event monitor tables and the threshold violation table

DELETE from ACTIVITY_DB2ACTIVITIES
DELETE from ACTIVITY_STMT_DB2ACTIVITIES
DELETE from THRESHOLDVIOLATIONS_THREVIO

CALL WLM_COLLECT_STATS()

**Exercise 4: Differentiating activities by activity type**

This exercise demonstrates how a work action set can be used to: Collect information about all activities of a certain type; apply a threshold to all activities of a certain type; isolate activities of a certain type by mapping them to a specific service subclass

Estimated time: 25-30 minutes

Work action sets are used to apply an action to an activity based on what the activity is doing rather than who submitted it (as is done with workloads).
Actions can be applied to either:

- All database activities of a certain type (using a database work action set).
- Only to activities of a certain type in a particular service class (using a service class work action set).

This exercise shows both methods.

Additional Information: There are other actions that can be applied, such as collecting statistics for activities of a certain type that are not covered in this exercise.

**Step 1: Create a work class set**

First, create a work class set containing work classes that will represent the specific types of activities you are interested in. This work class set will be used in conjunction with work action sets to perform actions on the selected types of activities. The following example creates a work class set containing work classes of all possible types, but if you were interested only in one activity type, your work class set could be created to only contain that one work class.

```
CREATE WORK CLASS SET all_class_types
  (WORK CLASS read_wc WORK TYPE READ,
   WORK CLASS write_wc WORK TYPE WRITE,
   WORK CLASS ddl_wc WORK TYPE DDL,
   WORK CLASS call_wc WORK TYPE CALL,
   WORK CLASS load_wc WORK TYPE LOAD,
   WORK CLASS all_wc WORK TYPE ALL POSITION LAST)
```

**Step 2: Enable the activities event monitor**

Enable the event monitor for activities that was created in Exercise 1.

```
SET EVENT MONITOR DB2ACTIVITIES STATE 1
```

**Step 3: Create a database work action set**

If you want to perform a particular action on all activities of a specific type (such as applying a threshold or collecting activity information), use a database work action set.

Create a work action set at the database level that contains work actions for the specific work class representing the type of activities you want isolated. For this example, we want to collect activity data for all DDL, READ and LOAD activities that run on the system and we also want to stop any large read activity from running. For this exercise, a large read activity is any select statement that has an estimated cost (in timerons) of greater than 10000.

```
CREATE WORK ACTION SET db_was FOR DATABASE
  USING WORK CLASS SET all_class_types
  (WORK ACTION collect_load_wa ON WORK CLASS load_wc
   COLLECT ACTIVITY DATA WITH DETAILS AND VALUES,
   WORK ACTION collect_ddl_wa ON WORK CLASS ddl_wc
   COLLECT ACTIVITY DATA WITH DETAILS AND VALUES,
   WORK ACTION collect_read_wa ON WORK CLASS read_wc
   COLLECT ACTIVITY DATA WITH DETAILS AND VALUES,
   WORK ACTION stop_large_read_wa on WORK CLASS read_wc
   WHEN ESTIMATEDSQLCOST > 10000 STOP EXECUTION )
```
Step 4: Run activities and view work action set statistics

Run the `work1.db2` and `work3.db2` scripts.
```
  db2 -o -tvf work1.db2
  db2 -o -tvf work3.db2
```

You can use the `WLM_GET_WORK_ACTION_SET_STATS` table function to access the work action set statistics in memory to get the number of times specific activity types have been run. Note that running the following query shows only the `load_wc`, `read_wc` and `ddl_wc` work classes since they are the only work classes that have an applicable work action. All the other activities are counted under the `*`:

```
CONNECT TO SAMPLE

SELECT SUBSTR(WORK_ACTION_SET_NAME, 1, 12) AS WORK_ACTION_SET_NAME,
  SUBSTR(WORK_CLASS_NAME, 1, 12) AS WORK_CLASS_NAME,
  LAST_RESET,
  SUBSTR(CHAR(ACT_TOTAL), 1, 10) AS TOTAL_ACTS
FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS('','-2')) AS WASSTATS
ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME
```

The output will look something like:

<table>
<thead>
<tr>
<th>WORK_ACTION_SET_NAME</th>
<th>WORK_CLASS_NAME</th>
<th>LAST_RESET</th>
<th>TOTAL_ACTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
4 record(s) selected.
```

Step 5: View the activity data collected

Information about every individual DDL, READ and LOAD activities was collected by the activities event monitor, due to the specification of the COLLECT ACTIVITY DATA work action that was applied to the `ddl_wc`, `read_wc`, and `load_wc` work classes in step 3. The following are a couple of examples of how you might want to look at this activity information.

To get some basic information about the activities, you can simply query the activity event monitor table with a statement such as the following:

```
SELECT ACTIVITY_ID,
  SUBSTR(ACTIVITY_TYPE, 1, 8) AS ACTIVITY_TYPE,
  VARCHAR(APPL_ID, 30) AS APPL_ID,
  VARCHAR(APPL_NAME, 10) AS APPL_NAME
FROM ACTIVITY_DB2ACTIVITIES
```

The output will look something like:

<table>
<thead>
<tr>
<th>ACTIVITY_ID</th>
<th>ACTIVITY_TYPE</th>
<th>APPL_ID</th>
<th>APPL_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

314  DB2 Workload Management Guide and Reference
To obtain additional information about each activity, such as activity text and what service class it ran under, you can perform a query similar to this one:

```
SELECT VARCHAR(A.APPL_NAME, 15) as APPL_NAME,
       VARCHAR(A.TPMON_CLIENT_APP, 20) AS CLIENT_APP_NAME,
       VARCHAR(A.APPL_ID, 30) as APPL_ID,
       VARCHAR(A.SERVICE_SUPERCLASS_NAME, 20) as SUPER_CLASS,
       VARCHAR(A.SERVICE_SUBCLASS_NAME, 20) as SUB_CLASS,
       SQLCODE,
       VARCHAR(S.STMT_TEXT, 300) AS STMT_TEXT
FROM ACTIVITY_DB2ACTIVITIES AS A, ACTIVITYSTMT_DB2ACTIVITIES AS S
WHERE A.APPL_ID = S.APPL_ID AND
  A.ACTIVITY_ID = S.ACTIVITY_ID AND
  A.UOW_ID = S.UOW_ID
```

The output will look something like:

<table>
<thead>
<tr>
<th>APPL_NAME</th>
<th>CLIENT_APP_NAME</th>
<th>APPL_ID</th>
<th>SUPER_CLASS</th>
<th>SUB_CLASS</th>
<th>SQLCODE</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2bp</td>
<td>CLP wasbsc.db2</td>
<td>*LOCAL.karenam.070815192410</td>
<td>SYSDEFAULTUS</td>
<td>ERCLASS</td>
<td>0 SELECT DISTINCT CURRENT SQLID FROM SYS IBM.SYSTABLES</td>
<td></td>
</tr>
<tr>
<td>db2bp</td>
<td>CLP work1.db2</td>
<td>*LOCAL.karenam.070815192418</td>
<td>SYSDEFAULTUS</td>
<td>ERCLASS</td>
<td>0 drop procedure stp2</td>
<td></td>
</tr>
</tbody>
</table>

26 record(s) selected.
Note that one of the activities has an SQLCODE of -4712. This indicates execution of the activity was stopped due to a threshold violation. The threshold defined for the stop_large_read_wa work action will prevent any SELECT statement with an estimated cost of greater than 10000 from executing.

Additional information: Load activities (not including load from a cursor) do not have an entry in the activity statement event monitor table (activitystmt_db2activities table) which explains why there is no record for the single load activity that is run by the work1.db2 script in the output shown previously in the last query. The reason for this is that load activities are not SQL statements. For load from cursor activities, there is an entry for the cursor statement in the activity statement event monitor table because the cursor itself is a separate activity. There is an entry for all load activities in the activities event monitor table (activity_db2activities).

**Step 6: Disable work actions**

Before moving on to the service class work action set, drop the database work action set.

```
DROP WORK ACTION SET db_was
```

Additional information: Before dropping any concurrency threshold, that threshold must first be disabled. In this case, there are no work actions that represent a concurrency threshold but if there were, the only way to disable it would be by disabling the work action. A work action threshold cannot be manipulated through THRESHOLD SQL statements; they can be manipulated only through WORK ACTION SET SQL statements. Only work actions that represent concurrency thresholds need to be disabled before dropping the subsequent work action set. For this exercise, because there are no work actions that represent a concurrency threshold, there is no need to disable any of the work actions before dropping the work action set.

If you want to apply a particular action, such as a threshold, to all the activities of a certain type running in a service super class, you should consider using a service class work action set. You can create a mapping work action to map specific types of activities to a specific service subclass and then apply a threshold to that service subclass. The following steps demonstrate how service class work action sets might be used.

**Step 7: Create a service class to and create a workload**

Create a service subclass under the work1_sc service super class that was created in Exercise 2 Step 2.

```
CREATE SERVICE CLASS work1_sc_read UNDER work1_sc
```

The service super class work1_sc is the service class that the activities will be mapped to through the workloads. The service subclass work1_sc_read is the service class that the read activities will be mapped to through the work action.
Create a workload so that all activities submitted by the work3.db2 script will be mapped to work1_sc service super class. Note that activities from work1.db2 are already being mapped to work1_sc from one of the previous exercises.

CREATE WORKLOAD work3_wl CURRENT CLIENT_APPLNAME('CLP work3.db2')
    SERVICE CLASS work1_sc

Step 8: Create a service class work action set

Create a work action set at the service class level that contains work actions that apply to the specific work classes representing the types of activities you want isolated. For this example, we want to collect activity data for all DDL, read, and load activities that run under the work1_sc service class and we also want to map read activities to a separate service subclass so that we can treat them differently; in this case, a threshold will be applied to the service subclass to stop any large SELECT statements from running.

CREATE WORK ACTION SET sc_was FOR SERVICE CLASS work1_sc
    USING WORK CLASS SET all_class_types ( WORK ACTION collect_load_wa ON WORK CLASS load_wc
        COLLECT ACTIVITY DATA ON ALL MEMBERS WITH DETAILS AND VALUES,
        WORK ACTION collect_ddl_wa ON WORK CLASS ddl_wc
        COLLECT ACTIVITY DATA ON ALL MEMBERS WITH DETAILS AND VALUES,
        WORK ACTION collect_read_wa ON WORK CLASS read_wc
        COLLECT ACTIVITY DATA ON ALL MEMBERS WITH DETAILS AND VALUES,
        WORK ACTION map_read_wa ON WORK CLASS read_wc
        MAP ACTIVITY TO work1_sc_read)

Step 9: Create a service class threshold

To get an effect similar to the stop_large_read_wa work action that prevented any large SELECT statements from running, create an ESTIMATEDSQLCOST threshold and apply it to the work1_sc_read service subclass.

CREATE THRESHOLD stop_large_activities FOR SERVICE CLASS work1_sc_read
    UNDER work1_sc
    ACTIVITIES ENFORCEMENT DATABASE
    WHEN ESTIMATEDSQLCOST >10000 STOP EXECUTION

Step 10: Clear the activity tables, reset the statistics, and run activities

Clear out all of the activity tables so that you can start afresh before running the script again. Then call the wlm_collect_stats() stored procedure to reset the statistics

DELETE FROM activity_db2activities
DELETE FROM activitystmt_db2activities
DELETE FROM activityvals_db2activities
CALL wlm_collect_stats()

Now, run work1.db2 and work3.db2 scripts once.

db2 -o -tvf work1.db2
db2 -o -tvf work3.db2

Note the SQL04712 error for activities that caused the threshold to be exceeded.

Step 11: View work action set statistics

Use the WLM_GET_WORK_ACTION_SET_STATS table function to access the work action set statistics in memory to get the number of times specific activity
types have been run. Note that running the following query shows only the load_wc, ddl_wc, and the read_wc work classes since they are the only three work classes that have a work action applied to them. All the other activities end up being counted under "*":

```sql
CONNECT TO SAMPLE

SELECT SUBSTR(WORK_ACTION_SET_NAME, 1, 12) AS WORK_ACTION_SET_NAME,
       SUBSTR(CHAR(MEMBER), 1, 4) AS MEMB,
       SUBSTR(WORK_CLASS_NAME, 1, 12) AS WORK_CLASS_NAME, LAST_RESET,
       SUBSTR(CHAR(ACT_TOTAL), 1, 10) AS TOTAL_ACTS
FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS('', -2)) AS WASSTATS
ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME, MEMB
```

This time, output will look something such as the following:

<table>
<thead>
<tr>
<th>WORK_ACTION_SET_NAME</th>
<th>MEMB</th>
<th>WORK_CLASS_NAME</th>
<th>LAST_RESET</th>
<th>TOTAL_ACTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_WAS</td>
<td>0</td>
<td>*</td>
<td>2007-08-15-19.02.54.597999</td>
<td>12</td>
</tr>
<tr>
<td>SC_WAS</td>
<td>0</td>
<td>DDL_WC</td>
<td>2007-08-15-19.02.54.597999</td>
<td>12</td>
</tr>
<tr>
<td>SC_WAS</td>
<td>0</td>
<td>LOAD_WC</td>
<td>2007-08-15-19.02.54.597999</td>
<td>1</td>
</tr>
<tr>
<td>SC_WAS</td>
<td>0</td>
<td>READ_WC</td>
<td>2007-08-15-19.02.54.597999</td>
<td>12</td>
</tr>
</tbody>
</table>

4 record(s) selected.

**Step 12: View the activity data collected**

Now query the activity tables again to get information about the individual activities. Note the service subclass that the activities were run under.

```sql
SELECT VARCHAR(A.APPL_NAME, 15) as APPL_NAME,
       VARCHAR(A.TPMON_CLIENT_APP, 20) AS CLIENT_APP_NAME,
       VARCHAR(A.APPL_ID, 30) as APPL_ID,
       VARCHAR(A.SERVICE_SUPERCLASS_NAME, 20) as SUPER_CLASS,
       VARCHAR(A.SERVICE_SUBCLASS_NAME, 20) as SUB_CLASS,
       SQLCODE,
       VARCHAR(S.STMT_TEXT, 300) AS STMT_TEXT
FROM ACTIVITY_DB2ACTIVITIES AS A, ACTIVITYSTMT_DB2ACTIVITIES AS S
WHERE A.APPL_ID = S.APPL_ID AND
  A.ACTIVITY_ID = S.ACTIVITY_ID AND
  A.UOW_ID = S.UOW_ID
```

The output will look something like:

<table>
<thead>
<tr>
<th>APPL_NAME</th>
<th>CLIENT_APP_NAME</th>
<th>APPL_ID</th>
<th>SUPER_CLASS</th>
<th>SUB_CLASS</th>
<th>SQLCODE</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2bp</td>
<td>CLP work1.db2</td>
<td>*LOCAL.karenam.070815195555</td>
<td>WORK1_SC</td>
<td>WORK1_SC_READ</td>
<td>0 values(current client_appname)</td>
<td></td>
</tr>
<tr>
<td>db2bp</td>
<td>CLP work1.db2</td>
<td>*LOCAL.karenam.070815195555</td>
<td>WORK1_SC</td>
<td>WORK1_SC_READ</td>
<td>0 select * from org</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>db2bp</td>
<td>CLP work1.db2</td>
<td>*LOCAL.karenam.070815195555</td>
<td>WORK1_SC</td>
<td>SYSDEFAULTS_SUBCLASS</td>
<td>0 drop procedure stp2</td>
<td></td>
</tr>
<tr>
<td>db2bp</td>
<td>CLP work3.db2</td>
<td>*LOCAL.karenam.070815195600</td>
<td>WORK1_SC</td>
<td>WORK1_SC_READ</td>
<td>-4712 select count(*) from syscat.tables, syscat.tables, syscat.tables, syscat.tables, syscat.tables</td>
<td></td>
</tr>
</tbody>
</table>
Note again, that one of the activities has a SQLCODE of -4712, this time because of the stop_large_activities service class threshold that was created in step 9 that was violated because the estimated cost for that select statement was too large. Also notice that all read activities are being run under the work1_sc_read service subclass.

**Step 13: Reset for next exercise**

Disable the even monitor, drop the service class threshold and drop the service class work action set.

```
SET EVENT MONITOR DB2ACTIVITIES STATE 0
DROP THRESHOLD STOP_LARGE_ACTIVITIES
ALTER WORK ACTION SET SC_WAS
ALTER WORK ACTION COLLECT LOAD_WA DISABLE
ALTER WORK ACTION COLLECT DDL_WA DISABLE
ALTER WORK ACTION COLLECT_READ_WA DISABLE
ALTER WORK ACTION MAP_READ_WA DISABLE;
DROP WORK ACTION SET SC_WAS
```

Clear out all of the activity tables so that you can start afresh, before running the script again.

```
DELETE FROM activity_db2activities
DELETE FROM activitystmt_db2activities
DELETE from activityvals_db2activities
```

Disable all of the workloads that have been created so that all activities will run under the default user workload and get mapped to the default service super class.

```
ALTER WORKLOAD work1_wl DISABLE
ALTER WORKLOAD work2_wl DISABLE
ALTER WORKLOAD work3_wl DISABLE
ALTER WORKLOAD workth_wl DISABLE
```

Call the `wlm_collect_stats()` stored procedure to reset the statistics.

```
CALL WLM_COLLECT_STATS()
```

**Exercise 5: Using histograms for service classes**

This exercise demonstrates how to use the COLLECT AGGREGATE ACTIVITY DATA BASE option on a service class to produce histograms of coordinator activity lifetimes, coordinator activity execution times, and coordinator activity queue times.

Estimated time: 25-30 minutes

These three histograms are useful for knowing more than just the average lifetime, execution time, or queue time of the activities run on the system, since they can be used to calculate standard deviations and can reveal outliers. For more information on histograms, see “Histograms in workload management” on page 258.

Histograms are accessed through the statistics event monitor. This exercise reuses the statistics event monitor created in Exercise 1 Step 1.

Additional Information: The statistics event monitor is a write-to-table event monitor and contains logical data groups. The first is the control logical data
group, which every event monitor has, and then there are the logical data groups that are specific to the statistics event monitor type. The specific logical data groups are:

- histogrambin for histogram information
- qstats for threshold queue statistics
- scstats service class statistics
- wcstats for work class statistics
- wlstats for workload statistics

**Step 1: Create views for viewing histogram statistics**

Create several views to make querying the HISTOGRAMBIN_DB2STATISTICS table easier. The first view lists all of the histogram types available. This exercise reports just the three basic types: lifetime, execution time and queue time.

```sql
CREATE VIEW HISTOGRAMTYPES AS
  SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) AS HISTOGRAM_TYPE
  FROM HISTOGRAMBIN_DB2STATISTICS
```

A second view makes it easier to find out which service classes are having histograms collected for them. The HISTOGRAMBIN_DB2STATISTICS table identifies the service classes for which histograms are being collected using the service class ID. Joining this table with the SERVICECLASSES catalog table permits the service class information to be presented with the service super class name and service subclass name instead of the service class ID.

```sql
CREATE VIEW HISTOGRAMSERVICECLASSES AS
  SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) AS HISTOGRAM_TYPE,
          SUBSTR(PARENTSERVICECLASSNAME,1,24) AS SERVICE_SUPERCLASS,
          SUBSTR(SERVICECLASSNAME,1,24) AS SERVICE_SUBCLASS
  FROM HISTOGRAMBIN_DB2STATISTICS AS H,
       SYSCAT.SERVICECLASSES AS S
  WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID
```

The third view lists all of the times that a histogram of a given type was collected for a given service class. Such as the histogramserviceclasses view, it also joins the HISTOGRAMBIN_DB2STATISTICS table with the SERVICECLASSES catalog table. The difference is that it includes the STATISTICS_TIMESTAMP column as one of the columns in the view.

```sql
CREATE VIEW HISTOGRAMTIMES AS
  SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) AS HISTOGRAM_TYPE,
          SUBSTR(PARENTSERVICECLASSNAME,1,24) AS SERVICE_SUPERCLASS,
          SUBSTR(SERVICECLASSNAME,1,24) AS SERVICE_SUBCLASS,
          STATISTICS_TIMESTAMP AS TIMESTAMP
  FROM HISTOGRAMBIN_DB2STATISTICS AS H,
       SYSCAT.SERVICECLASSES AS S
  WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID
```

The fourth and final view will be used to show the histograms themselves. It also demonstrates something that one often needs to do when dealing with histograms, which is to aggregate them over time. This view shows the top of each bin and the number of activities that were counted towards each bin. For the three histograms in this exercise, the BIN_TOP field measures the number of milliseconds in the activity lifetime, execution time or queue time. When BIN_TOP is, say 3000 milliseconds and the BIN_TOP of the previous bin is 2000 milliseconds and the NUMBER_IN_BIN is ten for a lifetime histogram, you know that ten activities had a lifetime that was between 2 and 3 seconds.
CREATE VIEW HISTOGRAMS(HISTOGRAM_TYPE, SERVICE_SUPERCLASS, SERVICE_SUBCLASS, BIN_TOP, NUMBER_IN_BIN) AS
SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) AS HISTOGRAM_TYPE, SUBSTR(PARENTSERVICECLASSNAME,1,24) AS SERVICE_SUPERCLASS, SUBSTR(SERVICECLASSNAME,1,24) AS SERVICE_SUBCLASS, TOP AS BIN_TOP, SUM(NUMBER_IN_BIN) AS NUMBER_IN_BIN
FROM HISTOGRAMBIN_DB2STATISTICS AS H, SYSCAT.SERVICECLASSES AS S
WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID
GROUP BY HISTOGRAM_TYPE, PARENTSERVICECLASSNAME, SERVICECLASSNAME, TOP

Step 2: Turn on the collection of histograms

The activity lifetime, queue time, and execution time histograms are collected for a service subclass when the base collect aggregate activity data option is enabled for the subclass. Enable the base aggregate activity data collection for the default subclass under the default user super class using the COLLECT AGGREGATE ACTIVITY DATA clause.

Note that all activities will be run in the default user service class since all the user defined workloads were disabled at the end of the previous exercise.

ALTER SERVICE CLASS SYSDEFAULTSUBCLASS
    UNDER SYSDEFAULTUSERCLASS
    COLLECT AGGREGATE ACTIVITY DATA BASE

Step 3: Activate the statistics event monitor

Activate the statistics event monitor that was created earlier so that it may receive the aggregate data whenever it is collected.

SET EVENT MONITOR DB2STATISTICS STATE 1

Step 4: Run activities and send statistics to the statistics event monitor

Now some activities can be run. After the activities have finished, the WLM_COLLECT_STATS stored procedure is called to send the statistics (including the activity lifetime, execution time and queue time histograms for the default user service class) to the active statistics event monitor. These histograms contain data about all activities that executed in the default user service class since aggregate activity statistics were enabled. Calling this stored procedure also resets the statistics. To show changes in database activity over time, three collection intervals are created. In the first interval, run two scripts, work1.db2 and work2.db2, and then collect and reset the statistics.

```
db2 -o -tvf work1.db2
db2 -o -tvf work2.db2
CONNECT TO SAMPLE
CALL WLM_COLLECT_STATS()
```

In the second interval, only run the work1.db2 script once and then collect and reset the statistics.
db2 -o -tvf work1.db2
CONNECT TO SAMPLE
CALL WLM_COLLECT_STATS()

In the third interval, run work1.db2 twice and run work2.db2 script once and then collect and reset the statistics.

db2 -o -tvf work1.db2
db2 -o -tvf work2.db2
db2 -o -tvf work1.db2
CONNECT TO SAMPLE
CALL WLM_COLLECT_STATS()

Collecting data periodically such as this permits you to watch how work on your system changes over time.

Additional Information: Collecting data periodically does not need to be a manual operation. Using the WLM_COLLECT_INT database configuration parameter, one can set the interval in minutes after which statistics collection and reset automatically occurs.

**Step 5: Query views to view statistics**

Now that statistics have been collected, the views created earlier can be used to look at the statistics. The HISTOGRAMTYPES view just returns the types of histograms available.

```
SELECT * FROM HISTOGRAMTYPES
```

<table>
<thead>
<tr>
<th>HISTOGRAM_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordActExecTime</td>
</tr>
<tr>
<td>CoordActLifetime</td>
</tr>
<tr>
<td>CoordActQueueTime</td>
</tr>
</tbody>
</table>

3 record(s) selected.

Since the BASE option was used when altering the service class, there are three histograms: lifetime, exectime and queuetime. The HISTOGRAMSERVICECLASSES view permits you to see the service classes for which a histogram was collected. The following example restricts the output to that of the CoordActLifetime histogram only. Since aggregate activity collection was only turned on for the default user service class’s default subclass, only that class is shown when selecting from the HISTOGRAMSERVICECLASSES view.

```
SELECT * FROM HISTOGRAMSERVICECLASSES
WHERE HISTOGRAM_TYPE = 'CoordActLifetime'
ORDER BY SERVICE_SUPERCLASS, SERVICE_SUBCLASS
```

<table>
<thead>
<tr>
<th>HISTOGRAM_TYPE</th>
<th>SERVICE_SUPERCLASS</th>
<th>SERVICE_SUBCLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoordActLifetime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
</tr>
</tbody>
</table>

1 record(s) selected.

The HISTOGRAMTIMES view shows the times when histograms were collected. Since the WLM_COLLECT_STATS procedure was run three times, there are three timestamps for the lifetime histogram shown.
SELECT * FROM HISTOGRAMTIMES
WHERE HISTOGRAM_TYPE = 'CoordActLifetime'
    AND SERVICE_SUPERCLASS = 'SYSDEFAULTUSERCLASS'
    AND SERVICE_SUBCLASS = 'SYSDEFAULTSUBCLASS'
ORDER BY TIMESTAMP

<table>
<thead>
<tr>
<th>HISTOGRAM_TYPE</th>
<th>SERVICE_SUPERCLASS</th>
<th>SERVICE_SUBCLASS</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordActLifetime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2007-08-05-20.44.51.519380</td>
</tr>
<tr>
<td>CoordActLifetime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2007-08-05-21.04.27.131281</td>
</tr>
<tr>
<td>CoordActLifetime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2007-08-05-21.08.27.474168</td>
</tr>
</tbody>
</table>

3 record(s) selected.

The last view, HISTOGRAMS, is for looking at the histograms themselves. Unlike the HISTOGRAMTIMES view that lists each collection interval as its own row, this view aggregates histogram data across multiple intervals to produce a single histogram of a given type for a given service class.

SELECT BIN_TOP, NUMBER_IN_BIN FROM HISTOGRAMS
WHERE HISTOGRAM_TYPE = 'CoordActLifetime'
    AND SERVICE_SUPERCLASS = 'SYSDEFAULTUSERCLASS'
    AND SERVICE_SUBCLASS = 'SYSDEFAULTSUBCLASS'
ORDER BY BIN_TOP

<table>
<thead>
<tr>
<th>BIN_TOP</th>
<th>NUMBER_IN_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>19</td>
<td>8</td>
</tr>
<tr>
<td>29</td>
<td>12</td>
</tr>
<tr>
<td>44</td>
<td>13</td>
</tr>
<tr>
<td>68</td>
<td>23</td>
</tr>
<tr>
<td>103</td>
<td>11</td>
</tr>
<tr>
<td>158</td>
<td>2</td>
</tr>
<tr>
<td>241</td>
<td>5</td>
</tr>
<tr>
<td>369</td>
<td>0</td>
</tr>
<tr>
<td>562</td>
<td>0</td>
</tr>
<tr>
<td>858</td>
<td>0</td>
</tr>
<tr>
<td>1309</td>
<td>0</td>
</tr>
<tr>
<td>1997</td>
<td>0</td>
</tr>
<tr>
<td>3046</td>
<td>0</td>
</tr>
<tr>
<td>4647</td>
<td>0</td>
</tr>
<tr>
<td>7089</td>
<td>0</td>
</tr>
<tr>
<td>10813</td>
<td>0</td>
</tr>
<tr>
<td>16493</td>
<td>0</td>
</tr>
<tr>
<td>25157</td>
<td>0</td>
</tr>
<tr>
<td>38373</td>
<td>0</td>
</tr>
<tr>
<td>58532</td>
<td>0</td>
</tr>
<tr>
<td>89280</td>
<td>0</td>
</tr>
<tr>
<td>136181</td>
<td>0</td>
</tr>
<tr>
<td>207720</td>
<td>0</td>
</tr>
<tr>
<td>316840</td>
<td>0</td>
</tr>
<tr>
<td>483283</td>
<td>0</td>
</tr>
<tr>
<td>737162</td>
<td>0</td>
</tr>
<tr>
<td>1124409</td>
<td>0</td>
</tr>
<tr>
<td>1715085</td>
<td>0</td>
</tr>
<tr>
<td>2616055</td>
<td>0</td>
</tr>
</tbody>
</table>
The output from the histograms can then be used as input into a graphing tool to generate a graph. The following diagram shows a graph that was created using a Ruby Graphing Library called Gruff Graphs.

Lifetime histogram for SYSDEFAULTUSERCLASS (CoordActLifetime):

Running the query of this view, should produce output that will not be exactly the same as what is shown in the preceding lifetime histogram graph since activity lifetimes depend on the performance of the system. In the previous output, there are 41 bins and all of the largest bins are empty. At the top, there is a bin whose BIN_TOP is -1. This bin represents all of those activities whose lifetime was too large to fit in the histogram. Seeing a NUMBER_OF_BIN greater than zero when the BIN_TOP is -1 indicates that you should probably increase the high bin value of your histogram. In the previous output, the NUMBER_IN_BIN is 0, so there is no need to make such a change. A large number of activities, 88 in this case, were counted in the bin with a BIN_TOP of 1. This is the lowest bin and it means that 88 activities had a lifetime between 0 and 1 milliseconds. Another piece of information that can be extracted from the histogram is that, since the largest BIN_TOP for which there is a corresponding non-zero NUMBER_IN_BIN is 241, the largest lifetime of any activity in the workloads collected in this histogram was between 158 milliseconds and 241 milliseconds. The COORD_ACT_LIFETIME_TOP column in the SCSTATS_DB2STATISTICS table gives a more precise measurement of the lifetime of the activity with the largest lifetime.

The same query can be repeated with a histogram_type of CoordActExecTime instead of CoordActLifetime. The execution time histogram is expected to be similar but not identical to the lifetime histogram. The reason they are different, even when there is no queuing, is that execution time does not include initialization time or cursor idle time, while lifetime does.
## Execution Time Histogram for SYSDEFAULTUSERCLASS (CoordActExecTime)

The execution time histogram for SYSDEFAULTUSERCLASS (CoordActExecTime) is as follows:

<table>
<thead>
<tr>
<th>BIN_TOP</th>
<th>NUMBER_IN_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>112</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>29</td>
<td>12</td>
</tr>
<tr>
<td>44</td>
<td>7</td>
</tr>
<tr>
<td>68</td>
<td>11</td>
</tr>
<tr>
<td>103</td>
<td>11</td>
</tr>
<tr>
<td>158</td>
<td>2</td>
</tr>
<tr>
<td>241</td>
<td>5</td>
</tr>
<tr>
<td>369</td>
<td>0</td>
</tr>
<tr>
<td>562</td>
<td>0</td>
</tr>
<tr>
<td>858</td>
<td>0</td>
</tr>
<tr>
<td>1309</td>
<td>0</td>
</tr>
<tr>
<td>1997</td>
<td>0</td>
</tr>
<tr>
<td>3046</td>
<td>0</td>
</tr>
<tr>
<td>4647</td>
<td>0</td>
</tr>
<tr>
<td>7089</td>
<td>0</td>
</tr>
<tr>
<td>10813</td>
<td>0</td>
</tr>
<tr>
<td>16493</td>
<td>0</td>
</tr>
<tr>
<td>25157</td>
<td>0</td>
</tr>
<tr>
<td>38373</td>
<td>0</td>
</tr>
<tr>
<td>58552</td>
<td>0</td>
</tr>
<tr>
<td>89280</td>
<td>0</td>
</tr>
<tr>
<td>136181</td>
<td>0</td>
</tr>
<tr>
<td>207720</td>
<td>0</td>
</tr>
<tr>
<td>316840</td>
<td>0</td>
</tr>
<tr>
<td>483283</td>
<td>0</td>
</tr>
<tr>
<td>737162</td>
<td>0</td>
</tr>
<tr>
<td>1124409</td>
<td>0</td>
</tr>
<tr>
<td>1715885</td>
<td>0</td>
</tr>
<tr>
<td>2616855</td>
<td>0</td>
</tr>
<tr>
<td>3990325</td>
<td>0</td>
</tr>
<tr>
<td>6086529</td>
<td>0</td>
</tr>
<tr>
<td>9283913</td>
<td>0</td>
</tr>
<tr>
<td>14160950</td>
<td>0</td>
</tr>
<tr>
<td>21600000</td>
<td>0</td>
</tr>
</tbody>
</table>

41 record(s) selected.
Once again, a large number of activities are counted in the first bin and the highest execution time of any activity is at most 241 milliseconds.

Finally, the HISTOGRAMS view will be used to look at the CoordActQueueTime histogram. This is the simplest histogram because there is no queuing, since no queuing thresholds were created or enabled in this exercise.

```
SELECT BIN_TOP, NUMBER_IN_BIN FROM HISTOGRAMS
WHERE HISTOGRAM_TYPE = 'CoordActQueueTime'
AND SERVICE_SUPERCLASS = 'SYSDEFAULTUSERCLASS'
AND SERVICE_SUBCLASS = 'SYSDEFAULTSUBCLASS'
ORDER BY BIN_TOP
```

<table>
<thead>
<tr>
<th>BIN_TOP</th>
<th>NUMBER_IN_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>177</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>0</td>
</tr>
<tr>
<td>158</td>
<td>0</td>
</tr>
<tr>
<td>241</td>
<td>0</td>
</tr>
<tr>
<td>369</td>
<td>0</td>
</tr>
<tr>
<td>562</td>
<td>0</td>
</tr>
<tr>
<td>858</td>
<td>0</td>
</tr>
<tr>
<td>1309</td>
<td>0</td>
</tr>
<tr>
<td>1997</td>
<td>0</td>
</tr>
<tr>
<td>3046</td>
<td>0</td>
</tr>
<tr>
<td>4647</td>
<td>0</td>
</tr>
<tr>
<td>7089</td>
<td>0</td>
</tr>
<tr>
<td>10813</td>
<td>0</td>
</tr>
<tr>
<td>16493</td>
<td>0</td>
</tr>
<tr>
<td>25157</td>
<td>0</td>
</tr>
<tr>
<td>38373</td>
<td>0</td>
</tr>
<tr>
<td>58532</td>
<td>0</td>
</tr>
<tr>
<td>89280</td>
<td>0</td>
</tr>
<tr>
<td>136181</td>
<td>0</td>
</tr>
</tbody>
</table>
Every activity was counted in the 0 to 1 millisecond bin because every activity spent zero milliseconds queuing.

The last several queries looked at activity lifetimes, execution times and queue times broken down into bins but aggregated across multiple intervals. The following query presents the same information from a different perspective. It shows averages instead of histograms and, rather than combining the intervals, it shows each interval individually. It also reports a count of the number of completed activities which shows how many activities completed in each interval. It uses the SCSTATS_DB2STATISTICS table instead of the HISTOGRAMBIN_DB2STATISTICS table.

```
SELECT STATISTICS_TIMESTAMP,
    COORD_ACT_LIFETIME_AVG AS LIFETIMEAVG,
    COORD_ACT_EXEC_TIME_AVG AS EXECTIMEAVG,
    COORD_ACT_QUEUE_TIME_AVG AS QUEUETIMEAVG,
    COORD_ACT_COMPLETED_TOTAL AS COMPLETED_TOTAL
FROM SCSTATS_DB2STATISTICS
WHERE SERVICE_SUPERCLASS_NAME = 'SYSDEFAULTUSERCLASS'
    AND SERVICE_SUBCLASS_NAME = 'SYSDEFAULTSUBCLASS'
ORDER BY STATISTICS_TIMESTAMP
```

<table>
<thead>
<tr>
<th>STATISTICS_TIMESTAMP</th>
<th>LIFETIMEAVG</th>
<th>EXECTIMEAVG</th>
<th>QUEUETIMEAVG</th>
<th>COMPLETED_TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-08-07-14.07.44.51153</td>
<td>508</td>
<td>475</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
The result shows that average lifetimes are slightly higher than average execution times for each interval and all three are just over a half a second or less. The average queue time, as expected, is zero. The counts of the number of completed activities in each interval is as expected because workloads 1 and 2 were run in the first interval which resulted in 77 activities collected, workload 1 ran alone in the second interval which resulted in 39 activities, and workload 1 ran twice and workload 2 ran once in the third interval, which resulted in 113 activities.

**Step 6: Reset for the next exercise**

The final step is to turn off collection of aggregate activities on the default user service class and drop the views and delete the information in the statistics tables.

```
ALTER SERVICE CLASS SYSDEFAULTSUBCLASS
    UNDER SYSDEFAULTUSERCLASS
    COLLECT AGGREGATE ACTIVITY DATA NONE

DROP VIEW histograms
DROP VIEW histogramtimes
DROP VIEW histogramserviceclasses
DROP VIEW histogramserviceclasses

SET EVENT MONITOR DB2STATISTICS STATE 0

DELETE FROM HISTOGRAMBIN_DB2STATISTICS
DELETE FROM SCSTATS_DB2STATISTICS
```

**Exercise 6: Investigating delays with WLM table functions**

This exercise demonstrates how you can determine the cause of an application slow down with the DB2 WLM monitoring facilities.

Estimated time: 10-15 minutes

The DB2 WLM monitoring facilities provide information and statistics for work in a database. Once the cause of a slow-down is identified, you can remedy the situation.

**Step 1: Run activities**

Two applications are used in this exercise, app1.db2 and app2.db2. Both applications perform DML operations on the SAMPLE database. Run the app1.db2 script in one window followed immediately by the app2.db2 script in a second window.

```
db2 -tvf app1.db2
db2 -tvf app2.db2
```

**Step 2: View currently active workload occurrences**

The app2.db2 script should now be hanging. From a third window, issue table function WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES to find the states of all applications running on the database. For this example, you can think
of a workload occurrence as the same as an application. This table function shows information for all workload occurrences in a service class. Since we want to see all workload occurrences in the database, we use wildcards represented by '' as service superclass name and service subclass name input parameters.

```sql
CONNECT TO SAMPLE

SELECT INTEGER(APPLICATION_HANDLE) APPL_HANDLE,
       VARCHAR(CLIENT_APPLNAME, 15) AS APPL_NAME,
       VARCHAR(SYSTEM_AUTH_ID, 20) AS USER_ID
FROM TABLE
   (WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', -2))
```

The output will look something such as the following:

```
APPL_HANDLE APPL_NAME USER_ID
----------- --------------- ----------------------
12 CLP app1.db2 DB2USR1
17 CLP app2.db2 DB2USR1
18 - DB2USR1
19 - DB2USR1
```

4 record(s) selected.

From the output, we can tell that the application handle for app2.db2 is 17.

**Step 3: Find the agent for the application**

To find out what the agents for app2.db2 are doing use the WLM_GET_SERVICE_CLASS_AGENTS table function. This table function shows information on agents working in a service class. Since we want to see the agents working for application handle 17, we specify this in the application_handle input parameter. For this example, we are not interested in agents for a particular service class, so we specify wildcards for the service superclass name and service subclass name input parameters.

```sql
SELECT INTEGER(APPLICATION_HANDLE) AS APPL_HANDLE,
       UOW_ID, ACTIVITY_ID,
       VARCHAR(AGENT_TYPE, 15) AS AGENT_TYPE,
       VARCHAR(AGENT_STATE, 10) AS AGENT_STATE,
       VARCHAR(EVENT_TYPE, 10) AS EVENT_TYPE,
       VARCHAR(EVENT_OBJECT, 10) AS EVENT_OBJ,
       VARCHAR(EVENT_STATE, 10) AS EVENT_STATE
FROM TABLE
   (WLM_GET_SERVICE_CLASS_AGENTS('', '', 17, -2))
```

The output will look something like

```
APPL_HANDLE UOW_ID ACTIVITY_ID AGENT_TYPE AGENT_STATE EVENT_TYPE EVENT_OBJ EVENT_STATE
----------- ----------- ----------- --------------- ----------- ----------
17 1 2 COORDINATOR ACTIVE ACQUIRE
```

1 record(s) selected.

From the output, you can see that the coordinator agent for application 17 is idle and waiting to acquire a lock. This is the reason why app2.db2 appears to be hanging.
Step 4: Find the problem application and resolve the problem

Now that we know why the application is hanging, we can remedy the situation. We know the application is waiting on a lock. To find out which lock this application is waiting on and which application is holding the lock, we can use the db2pd tool. First, we need to find out the current transaction number for our hanging application: Issue db2pd -transactions for application handle 17.

db2pd -db sample -transactions app=17

The output will look something such as the following:

<table>
<thead>
<tr>
<th>Address</th>
<th>AppHandl [nod-index]</th>
<th>TranHdl</th>
<th>Locks</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tflag</td>
<td>Tflag2</td>
<td>Firstlsn</td>
<td>Lastlsn</td>
</tr>
<tr>
<td></td>
<td>SpaceReserved</td>
<td>TID</td>
<td>AxRegCnt</td>
<td>GXID</td>
</tr>
<tr>
<td>0x07000000302A7080</td>
<td>17</td>
<td>[000-00017]</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x000000000000</td>
<td>0x000000000000</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0x000000000AC3</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

From the output, we can tell that application 17 has transaction handle 7. We can now find which locks this transaction is waiting on by issuing the db2pd -locks command for transaction handle 7.

db2pd -db sample -locks 7 wait

The output will look something such as the following:

<table>
<thead>
<tr>
<th>Address</th>
<th>TranHdl</th>
<th>Lockname</th>
<th>Type</th>
<th>Mode</th>
<th>Sts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>Dur HoldCount</td>
<td>Att</td>
<td>ReleaseFlg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x07000000304013F0</td>
<td>7</td>
<td>00020010000000000640002D52</td>
<td>Row</td>
<td>.NS</td>
<td>W</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0x00000002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The output shows that the application is waiting on a row lock. The owner of the lock has transaction handle 2. This transaction is holding the lock and causing our hang. The final step is to determine the corresponding application handle for transaction handle 2. Issue db2pd -transactions command for transaction handle 2.

db2pd -db sample -transactions 2

The output will look something such as the following:

<table>
<thead>
<tr>
<th>Address</th>
<th>AppHandl [nod-index]</th>
<th>TranHdl</th>
<th>Locks</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tflag</td>
<td>Tflag2</td>
<td>Firstlsn</td>
<td>Lastlsn</td>
</tr>
<tr>
<td></td>
<td>SpaceReserved</td>
<td>TID</td>
<td>AxRegCnt</td>
<td>GXID</td>
</tr>
<tr>
<td>0x07000000302A2080</td>
<td>12</td>
<td>[000-00012]</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x000000000000</td>
<td>0x000000000000</td>
<td>0</td>
</tr>
<tr>
<td>396</td>
<td>0x000000000AB8</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

From the output, we can see that transaction handle 2 corresponds to application handle 12. Referring back to the results from table function WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES, you can see that application 12 refers to app1.db2. This application is holding a row lock that is needed by app2.db2. To make app2.db2 proceed, you may commit, rollback or terminate the unit of work or process from the window running app1.db2. Alternatively, you may also force off app1.db2 by issuing FORCE APPLICATION on application handle 12.

db2 force application (12)

Additional Information: Another way to diagnose hanging applications due to lock contention is to use the MON_GET_APPL_LOCKWAIT and the MON_FORMAT_LOCK_NAME monitor table functions. These table functions provide information on lock holders and waiters.
Exercise 7: Cancelling an ongoing activity

This exercise demonstrates how to cancel an activity that is currently active using the WLM_CANCEL_ACTIVITY procedure.

Estimated time: 5-10 minutes

**Step 1: Issue a long running query**

From a CLP window, run the following script that issues a long running query
```
db2 -tvf longquery.db2
```

**Step 2: Get the application handle**

From another CLP window, call the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES to get the application handle, unit of work ID and activity ID of the cursor activity.

```
SELECT T.APPLICATION_HANDLE, T.UOW_ID, T.ACTIVITY_ID, T.ACTIVITY_TYPE
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES
           (CAST(NULL AS BIGINT), -2)) T
WHERE T.CLIENT_APPLNAME = 'CLP longquery.db2';
```

By joining the result of the table function with the APPLICATIONS administrative view, we can find the cursor activity that is run from within longquery.db2. The output would look something such as the following:

```
APPLICATION_HANDLE UOW_ID ACTIVITY_ID ACTIVITY_TYPE
-------------------- ----------- ----------- ----------------------------
----- 267 1 1 READ_DML
```

1 record(s) selected.

**Step 3: Cancel the activity**

From the same CLP window, call the WLM_CANCEL_ACTIVITY stored procedure to cancel the cursor activity obtained previously, using the application handle, unit of work ID, and activity ID obtained from the previous step:
```
CONNECT TO SAMPLE
CALL WLM_CANCEL_ACTIVITY (267, 1, 1)
CONNECT RESET
```

Note that in your case, the application handle, unit of work ID, and activity ID will be different.

In the first CLP window, you will see the following output returned by the long running query issued by longquery.db2.

```
SQL4725N The activity has been cancelled. SQLSTATE=57014
```

Chapter 6. Tutorial for DB2 workload management  331
Exercise 8: Discovering what types of activities are running on your system

This exercise demonstrates how you can use the DB2 workload management monitoring table functions and work action sets to discover what types of activities are running on your system.

Estimated time: 15-20 minutes

You might want to know the number of large activities or load utilities that are being run concurrently on your system, for example. Understanding the types of work being run on the system is important as different types of work will have different resource requirements and impacts on system performance.

**Step 1: Determining the number of activities of each type that are running on your system**

Before starting, you might want to show the number of activities of a certain type that are currently running by using the
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function:

```sql
CONNECT TO SAMPLE
SELECT ACTIVITY_TYPE,
       COUNT(*) AS NUMBER_RUNNING
FROM TABLE (WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(CAST(NULL AS BIGINT), -2)) AS T
GROUP BY ACTIVITY_TYPE
```

The output from this query will look something like:

<table>
<thead>
<tr>
<th>ACTIVITY_TYPE</th>
<th>NUMBER_RUNNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_DML</td>
<td>1</td>
</tr>
</tbody>
</table>

To get information about the different types of activities that have run on your system over a given period of time, you can use work class sets and work actions.

**Step 2: Create a database work action set with count activity work actions**

To count the number of times an activity of a specific type has been run over a period of time, a work action set needs to be created. In this example, because we are interested in the activities that are run on the entire system, the work action set will be created at the database level and is associated with the all_class_types work class set that was created in Exercise 4 Step 1. This work class set contains work classes for all types of recognized activities. If we were only interested in the activities being run in a specific service class, we would create a work action set at the service class level. For this example, we are also interested in the information for all types of activities so that the work action set contains a COUNT ACTIVITY work action for each work class in the all_class_types work class set.

```sql
CREATE WORK ACTION SET work1_was FOR DATABASE
   USING WORK CLASS SET all_class_types
   (WORK ACTION count_read_wa ON WORK CLASS read_wc COUNT ACTIVITY,
    WORK ACTION count_write_wa ON WORK CLASS write_wc COUNT ACTIVITY,
    WORK ACTION count_ddl_wa ON WORK CLASS ddl_wc COUNT ACTIVITY,
```

DB2 Workload Management Guide and Reference
WORK ACTION count_call_wa ON WORK CLASS call_wc COUNT ACTIVITY,
WORK ACTION count_load_wa ON WORK CLASS load_wc COUNT ACTIVITY,
WORK ACTION count_all_wa on WORK CLASS all_wc COUNT ACTIVITY)

Additional information: Each time an activity corresponding to a work class has
one or more work actions applied to it, a counter for the work class is incremented
by one. The COUNT ACTIVITY work action provides an efficient way to ensure
that the counter is updated. If you do not want to perform any other action on an
activity other than counting the number of activities of that type that have been
run, the COUNT ACTIVITY work action is the best approach.

**Step 3: Run some activities**

Run the work1.db2 script once.

db2 -tvf work1.db2

**Step 4: View work action set statistics**

You can use the WLM_GET_WORK_ACTION_SET_STATS table function to access
the work action set statistics in memory to get the number of times specific activity
types have been run. For example, the following query will tell you the number of
activities that were assigned to each of the work classes in the work class set that
has a work action associated with it:

CONNECT TO SAMPLE

```
SELECT SUBSTR(WORK_ACTION_SET_NAME, 1, 12) AS WORK_ACTION_SET_NAME,
       SUBSTR(WORK_CLASS_NAME, 1, 12) AS WORK_CLASS_NAME,
       LAST_RESET,
       SUBSTR(CHAR(ACT_TOTAL), 1, 12) AS TOTAL_ACTS
FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS('', -2)) AS WASSTATS
ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME, LAST_RESET
```

Additional Information: The blank included with the statement means that result is
not to be restricted by the argument (in this example, we want the information for
all of the work action sets). The value of the last argument, member, is the
wildcard character -2, which means that data from all database members is to be
returned.

The output from this query will look something like the following where "*"
represents all activities that do not fall into any of the defined work classes or that
fall into work classes with no work actions.

```
WORK_ACTION_SET_NAME WORK_CLASS_NAME LAST_RESET TOTAL_ACTS
------------------------------ -------------------------- ----------
WORK1_WAS * 2007-08-14-13.55.30.725886 0
WORK1_WAS ALL_WC 2007-08-14-13.55.30.725886 2
WORK1_WAS CALL_WC 2007-08-14-13.55.30.725886 4
WORK1_WAS DDL_WC 2007-08-14-13.55.30.725886 12
WORK1_WAS LOAD_WC 2007-08-14-13.55.30.725886 1
WORK1_WAS READ_WC 2007-08-14-13.55.30.725886 12
WORK1_WAS WRITE_WC 2007-08-14-13.55.30.725886 6
```

7 record(s) selected.
Step 5: Differentiate activities by more than their type and other attributes

You can separate out activities by more than just their types. For example, you might want to know how many large queries are being run.

Alter the work class set to add a new read work class that will represent large queries. For this example, a large query is any query that has a cardinality greater than 40.

```
ALTER WORK CLASS SET all_class_types
   ADD WORK CLASS large_wc WORK TYPE READ FOR CARDINALITY FROM 41 POSITION AT 1
```

Additional Information: Note that we positioned this work class at position 1. If the POSITION AT clause is not specified, the work class is positioned at the bottom of the work class set. When deciding which work class an activity belongs to, the work classes are checked in the order they are positioned and the first work class whose attributes match the activities attributes is the class that the activity gets assigned to. In this case, if the large_wc were positioned at the end of the list, the large activities would have been assigned to the read_wc since it was positioned ahead of large_wc.

Alter the work action set to add a COUNT ACTIVITY work action and apply it to the new work class.

```
ALTER WORK ACTION SET work1_was
   ADD WORK ACTION count_large_reads ON WORK CLASS large_wc COUNT ACTIVITY
```

Step 6: Reset the statistics and run some activities

Call the WLM_COLLECT_STATS stored procedure to reset the statistics that are stored in memory so that you are starting fresh and when you chose to query that workload management statistical information that is stored in memory, it will contain information for the activities that have been run from this point on.

```
CALL WLM_COLLECT_STATS()
```

Run the `work1.db2` script once.

```
db2 -tvf work1.db2
```

Step 7: View work action set statistics

Use the WLM_GET_WORK_ACTION_SET_STATS table function again to access the work action set statistics in memory to get the number of times specific activity types have been run.

```
CONNECT TO SAMPLE

SELECT SUBSTR(WORK_ACTION_SET_NAME, 1, 12) AS WORK_ACTION_SET_NAME,
   SUBSTR(MEMBER, 1, 4) AS MEMB,
   SUBSTR(WORK_CLASS_NAME, 1, 12) AS WORK_CLASS_NAME,
   LAST_RESET,
   SUBSTR(CHAR(ACT_TOTAL), 1, 12) AS TOTAL_ACTS
FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS('', -2)) AS WASSTATS
ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME, MEMB
```

The output will look something such as the following:

```
WORK_ACTION_SET_NAME MEMB WORK_CLASS_NAME LAST_RESET TOTAL_ACTS
--------------------- ---- --------------- -------------------------- ----
```

---------  ----  ---------------  --------------------------  ----

---------
WORK1_WAS  0  *  2007-08-14-13.55.35.650685  0
WORK1_WAS  0  ALL_WC  2007-08-14-13.55.35.650685  2
WORK1_WAS  0  CALL_WC  2007-08-14-13.55.35.650685  4
WORK1_WAS  0  DDL_WC  2007-08-14-13.55.35.650685  12
WORK1_WAS  0  LARGE_WC  2007-08-14-13.55.35.650685  4
WORK1_WAS  0  LOAD_WC  2007-08-14-13.55.35.650685  1
WORK1_WAS  0  READ_WC  2007-08-14-13.55.35.650685  8
WORK1_WAS  0  WRITE_WC  2007-08-14-13.55.35.650685  6

8 record(s) selected.

Note that this time four of the activities from the script are considered large activities.

Step 8: Reset for the next exercise

Drop the work action set:

```
ALTER WORK ACTION SET WORK1_WAS
ALTER WORK ACTION COUNT_READ_WA DISABLE
ALTER WORK ACTION COUNT_WRITE_WA DISABLE
ALTER WORK ACTION COUNT_DDL_WA DISABLE
ALTER WORK ACTION COUNT_CALL_WA DISABLE
ALTER WORK ACTION COUNT_LOAD_WA DISABLE
ALTER WORK ACTION COUNT_ALL_WA DISABLE
ALTER WORK ACTION COUNT_LARGE_READS DISABLE;
```

Drop work action set:
```
DROP WORK ACTION SET WORK1_WAS;
```

Exercise 9: Capturing detailed information about an executing activity

This exercise demonstrates how you can use the

```
WLM_CAPTURE_ACTIVITY_IN_PROGRESS
```

procedure to capture detailed information about a currently executing activity for later historical analysis.

Estimated time: 5-10 minutes

Activity information you capture is sent to the active event monitor for activities. Previous tasks showed how the COLLECT ACTIVITY DATA clause is used for workloads, service classes, work actions and thresholds to capture detailed activity information. This clause needs to be specified in advance before an activity begins executing and information is sent to the activities event monitor when the activity completes. The WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure permits you to capture information reactively when you notice a problem with an activity already in progress. When this procedure is used, information about an activity is sent to the activities event monitor immediately. Both basic and statement activity information are collected, but not input data.

Step 1: Enable activities event monitor

Enable the existing event monitor for activities you created in Exercise 1.
```
CONNECT TO SAMPLE
SET EVENT MONITOR DB2ACTIVITIES STATE 1
```

Step 2: Issue a long running query

From the CLP, run the following script that issues a long running query with a problematic cursor:
```
db2 -tvf longquery.db2
```
Step 3: Get the application handle

From a second CLP window, call
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES to obtain the application
handle, unit of work ID and activity ID for the activity that is run from within
longquery.db2.

CONNECT TO SAMPLE

SELECT T.APPLICATION_HANDLE, T.UOW_ID, T.ACTIVITY_ID, T.ACTIVITY_TYPE
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES
(CAST(NULL AS BIGINT), -2)) T
WHERE T.CLIENT_APPLNAME = 'CLP longquery.db2';

The output looks such as the following:

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>ACTIVITY_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------------------</td>
<td>--------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>267</td>
<td>1</td>
<td>1</td>
<td>READ_DML</td>
</tr>
</tbody>
</table>

1 record(s) selected.

Step 4: Capture information about the activity

From the same CLP window, call the WLM_CAPTURE_ACTIVITY_IN_PROGRESS
stored procedure using the application handle, unit of work ID, and activity ID
obtained from the previous step:

CONNECT TO SAMPLE

CALL WLM_CAPTURE_ACTIVITY_IN_PROGRESS (267, 1, 1)

CONNECT RESET

This step sends information about the activity to the active event monitor for
activities. Note that in your case, the application handle, unit of work ID, and
activity ID you specify may be different.

Step 5: View the activity information

Look at the information that was collected for the activity using a statement such
as the following:

SELECT VARCHAR(A.APPL_NAME, 15) as APPL_NAME,
    VARCHAR(A.TPMON_CLIENT_APP, 20) AS CLIENT_APP_NAME,
    VARCHAR(A.APPL_ID, 30) as APPL_ID,
    A.ACTIVITY_ID,
    A.UOW_ID,
    A.PARTIAL_RECORD,
    A.TIME_STARTED,
    A.TIME_COMPLETED,
    VARCHAR(S.STMT_TEXT, 300) AS STMT_TEXT
FROM ACTIVITY_DB2ACTIVITIES AS A,
     ACTIVITYSTMT_DB2ACTIVITIES AS S
WHERE A.APPL_ID = S.APPL_ID AND
    A.ACTIVITY_ID = S.ACTIVITY_ID AND
    A.UOW_ID = S.UOW_ID

The output looks something such as this:

<table>
<thead>
<tr>
<th>APPL_NAME</th>
<th>CLIENT_APP_NAME</th>
<th>APPL_ID</th>
<th>ACTIVITY_ID</th>
<th>UOW_ID</th>
<th>PARTIAL_RECORD</th>
<th>TIME_STARTED</th>
<th>TIME_COMPLETED</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note: Activities that are captured using the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure contain somewhat less information than if the COLLECT ACTIVITY DATA clause were used, because the activity is captured before it has completed execution. In particular, fields such as the completion timestamp (which shows only zeros) and the sqlcode do not apply. You can determine if an activity was collected using the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure by looking at the PARTIAL_RECORD column in the ACTIVITY table. If the PARTIAL_RECORD column has a value of 1 (as shown in the previous output), activity information was collected using WLM_CAPTURE_ACTIVITY_IN_PROGRESS. If the PARTIAL_RECORD column has a value of 0, activity information was collected after completion with the COLLECT ACTIVITY DATA clause.

Exercise 10: Generating historical data and reports

This exercise demonstrates how to use the WLM Historical Analysis Tool sample.

Estimated time: 20-25 minutes

DB2 database products include Perl sample scripts that provide information captured by the WLM activities event monitor for historical analysis. You can review or even modify these scripts to produce additional historical analysis reports to suit your needs. The perl scripts are:
- wlmhist.pl: generates historical data
- wlmhistrep.pl: produces reports from the historical data.

Step 1: Create the explain tables

In order to generate some historical data, the explain tables must exist under the schema of the user running the tool. To create the explain tables, go to the /sqllib/misc directory and run the following:

db2 connect to sample
db2 -tvf EXPLAIN.DDL

Step 2: Alter the service class to collect activity data

Enable activity collection by specifying the COLLECT ACTIVITY DATA clause on the WLM object of interest. For this exercise, we want to generate historical data for activities run in the default service subclass of the default user service superclass:

ALTER SERVICE CLASS SYSDEFAULTSUBCLASS UNDER SYSDEFAULTUSERCLASS
COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
Step 3: Enable the activities event monitor

Since the activities event monitor was created in Exercise 1 Step 1, enable it now if it is not enabled already.

SET EVENT MONITOR DB2ACTIVITIES STATE 1

Step 4: Run some activities

Run some activities so that activity data is collected to generate historical data on.

db2 -tvf work1.db2

db2 -tvf work2.db2

Step 5: Disable the event monitor for activities

It is highly recommended that you turn off the event monitor for activities before generating historical data. If you do not do this, any DML activities that are run as a result of the historical data generator may also be captured and put into the DB2 event monitor activity tables, thereby dramatically increasing the number of actual activities for which activity data is generated.

CONNECT TO SAMPLE

SET EVENT MONITOR DB2ACTIVITIES STATE 0

Step 6: Generate historical data

Run the historical data generator script, wlmhist.pl, to generate historical data for activities that are captured in the activities event monitor tables. The format is as follows:

```
perl wlmhist.pl dbname user password [fromTime toTime workloadid serviceClassName serviceSubclassName activityTable activityStmtTable]
```

Use a dash (-) to bypass any optional parameters.

Additional Information: The historical data generator (wlmhist.pl) script will generate only historical data for DML. If you have previously run the historical data generator (wlmhist.pl) script once or more, it is recommended that, before running it again, you clear the activityTable and activityStmtTable tables in order to avoid duplicating data. If you choose not to clear these two tables, be sure to use the fromTime and toTime input parameters to ensure you do not generate historical data for activities that have already had data generated for them.

For this exercise, generate historical data for all activities that have been captured in the activities event monitor.

```
perl wlmhist.pl sample db2inst1 password
```

You may notice some errors similar to the following:

```
Error running explain [IBM][CLI Driver][DB2/LINUXX8664] SQL0418N A statement contains a use of a parameter marker that is not valid. SQLSTATE=42610 for statement VALUES (TABLE_SCHEMA(:H00002, :H00003)) INTO :H00007
```

```
DB2::DB2::db do failed: [IBM][CLI Driver][DB2/LINUXX8664] SQL0418N A statement contains a use of a parameter marker that is not valid. SQLSTATE=42610
```
When generating historical data, explain is run on the actual statement. In some cases, explain cannot be run on some statements with parameter markers and an error is returned. Any activity that shows such an error will not have historical data generated for it.

Once the tool has completed generating historical data, it will tell you how many activities it has successfully generated historical data for.

**Step 7: Generate historical data reports**

Run the historical data report script `wlmhistrep.pl` to generate reports based on the data that was generated in step 1. The format is as follows:

```
wlmhistrep.pl dbAlias userId passwd [outputFile report schemaName fromTime toTime submitter]
```

Use a dash (-) to bypass optional parameters.

The `report` parameter can be any combination from the following letters:
- A: Tables hit
- B: Tables not hit
- C: Indexes hit
- D: Indexes not hit
- E: Submitters

If the `userId` parameter you specify is not the same as what was used to run the `wlmhist.pl` script when the `wlmhist` table was created, you must specify the correct `schemaName`. The `fromTime` and `toTime` parameters must be specified in timestamp format (for example 2007-06-06-17.00.00).

For this exercise, generate reports for tables hit and indexes not hit:

```
Perl wlmhistrep.pl sample db2inst1 password - AD
```

The output will look something such as the following:

### TABLES HIT REPORT FOR DATABASE sample

<table>
<thead>
<tr>
<th>TABLE NAME</th>
<th>TABLE SCHEMA</th>
<th>% HITS</th>
<th>TOTAL HITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE</td>
<td>KARENAM</td>
<td>7.14285714</td>
<td>2</td>
</tr>
<tr>
<td>INVENTORY</td>
<td>KARENAM</td>
<td>14.28571429</td>
<td>4</td>
</tr>
<tr>
<td>ORG</td>
<td>KARENAM</td>
<td>28.57142857</td>
<td>8</td>
</tr>
<tr>
<td>SALES</td>
<td>KARENAM</td>
<td>14.28571429</td>
<td>4</td>
</tr>
<tr>
<td>SYSROUTINES</td>
<td>SYSTOOLS</td>
<td>7.14285714</td>
<td>2</td>
</tr>
<tr>
<td>SYSTABLES</td>
<td>SYSTOOLS</td>
<td>21.42857143</td>
<td>6</td>
</tr>
<tr>
<td>SYSTABLESPACES</td>
<td>SYSTOOLS</td>
<td>7.14285714</td>
<td>2</td>
</tr>
</tbody>
</table>

### INDEXES NOT HIT REPORT FOR DATABASE sample

<table>
<thead>
<tr>
<th>TABLE NAME</th>
<th>TABLE SCHEMA</th>
<th>INDEX NAME</th>
<th>INDEX SCHEMA</th>
<th>INDEX TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN_ARGUMENT</td>
<td>KARENAM</td>
<td>ARG_11</td>
<td>KARENAM</td>
<td>REG</td>
</tr>
<tr>
<td>HMON_ATM_INFO</td>
<td>SYSTOOLS</td>
<td>ATM_UNIQ</td>
<td>SYSTOOLS</td>
<td>REG</td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>KARENAM</td>
<td>CUSTOM_C_ID_XMLIDX</td>
<td>KARENAM</td>
<td>XVIL</td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>KARENAM</td>
<td>CUSTOM_NAME_XMLIDX</td>
<td>KARENAM</td>
<td>XVIL</td>
</tr>
</tbody>
</table>
Exercise 11: Using extended aggregates for service classes

This exercise demonstrates how to use the COLLECT AGGREGATE ACTIVITY DATA EXTENDED option on a service class to produce histograms of coordinator activity inter-arrival times and estimated costs.

Estimated time: 25-30 minutes

The inter-arrival time is the time interval between the arrival of one activity into the system and the arrival of the next activity. The estimated cost of an activity represents an estimate by the SQL compiler of the system resources that will be used during the execution of the activity, and only applies to DML activities.

Inter-arrival time histograms or estimated cost histograms can be correlated with one another, with lifetime histograms, or with other lifetime statistics to determine whether changes in lifetime histograms or lifetime statistics could be due to one of the following events:

- A change in the complexity of the workload, such as a change in the distribution of the estimated costs
- A change in the activity arrival rate, determined from the inter-arrival time distribution
- A change in the system itself, such as the introduction of a new threshold, a change in the priority given to a service class, or a change in hardware

For more information on histograms, see “Histograms in workload management” on page 258.

Histograms are accessed through the statistics event monitor. This exercise reuses the statistics event monitor created in Exercise 1 Step 1.

Step 1: Create views for viewing histogram statistics

Create several views to make querying the HISTOGRAMBIN_DB2STATISTICS table easier. The first view lists all of the histogram types available. In this exercise, it reports just the three basic types: lifetime, execution time and queue time.
CREATE VIEW HISTOGRAMTYPES AS
    SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) HISTOGRAM_TYPE
    FROM HISTOGRAMBIN_DB2STATISTICS

A second view makes it easier to find out what service classes are having histograms collected for them. The HISTOGRAMBIN_DB2STATISTICS table reports the service class for which the histogram is being collected by giving the service class ID. Joining this table with the SERVICECLASSES catalog table will permit the service class information to be presented with the service class name and service subclass name instead of the service class ID.

CREATE VIEW HISTOGRAMSERVICECLASSES AS
    SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) HISTOGRAM_TYPE,
                SUBSTR(PARENTSERVICECLASSNAME,1,24) SERVICE_SUPERCLASS,
                SUBSTR(SERVICECLASSNAME,1,24) SERVICE_SUBCLASS
    FROM HISTOGRAMBIN_DB2STATISTICS H,
         SYSCAT.SERVICECLASSES S
    WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID

The third view lists all of the times that a histogram of a given type was collected for a given service class. Such as the HISTOGRAMSERVICECLASSES view, it joins the HISTOGRAMBIN_DB2STATISTICS table with the SERVICECLASSES catalog table. The difference is in the STATISTICS_TIMESTAMP column which is included as one of the columns in this view.

CREATE VIEW HISTOGRAMTIMES AS
    SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) HISTOGRAM_TYPE,
                SUBSTR(PARENTSERVICECLASSNAME,1,24) SERVICE_SUPERCLASS,
                SUBSTR(SERVICECLASSNAME,1,24) SERVICE_SUBCLASS,
                STATISTICS_TIMESTAMP TIMESTAMP
    FROM HISTOGRAMBIN_DB2STATISTICS H,
         SYSCAT.SERVICECLASSES S
    WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID

The fourth and final view will be used to show the histograms themselves. It also demonstrates a common task when dealing with histograms, which is to aggregate them over time. This view shows the top of each bin and the number of activities that were counted towards each bin. Of the two following histograms, the BIN_TOP field measures the number of milliseconds in the activity inter-arrival time and the number of timerons in the estimated cost. When BIN_TOP is 3000 milliseconds and the BIN_TOP of the previous bin is 2000 milliseconds and the NUMBER_IN_BIN is ten for an inter-arrival time histogram you know that there were ten activities which each arrived into the system between 2 and 3 seconds after the arrival of the previous activity, for example.

CREATE VIEW HISTOGRAMS(HISTOGRAM_TYPE, SERVICE_SUPERCLASS,
                        SERVICE_SUBCLASS, BIN_TOP, NUMBER_IN_BIN) AS
    SELECT DISTINCT SUBSTR(HISTOGRAM_TYPE,1,24) HISTOGRAM_TYPE,
                SUBSTR(PARENTSERVICECLASSNAME,1,24) SERVICE_SUPERCLASS,
                SUBSTR(SERVICECLASSNAME,1,24) SERVICE_SUBCLASS,
                TOP AS BIN_TOP,
                SUM(NUMBER_IN_BIN) AS NUMBER_IN_BIN
    FROM HISTOGRAMBIN_DB2STATISTICS H,
         SYSCAT.SERVICECLASSES S
    WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID
    GROUP BY HISTOGRAM_TYPE, PARENTSERVICECLASSNAME, SERVICECLASSNAME, TOP

**Step 2: Turn on collection of histograms**

Turning on the collection of histograms is done for the default user service class by altering its default subclass to collect aggregate activity data with the EXTENDED option. This provides both the three histograms available in the BASE option
(lifetime, execution time, and queue time) and the two histograms available only when using the EXTENDED option (inter-arrival time and estimated cost).

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

**Step 3: Activate statistics event monitor**

If not already active, activate the event monitor that was created earlier so that it can receive aggregate data whenever it is collected.

```
SET EVENT MONITOR DB2STATISTICS STATE 1
```

**Step 4: Run activities and send statistics to statistics event monitor**

First run some activities; after the activities have finished the WLM_COLLECT_STATS stored procedure is called to send the service class statistics to the active statistics event monitor (including the activity lifetime, execution time, queue time, inter-arrival time and estimated cost histograms for the default user service class). These histograms contain data about all activities that executed in the default user service class since aggregate activity statistics were enabled. Calling the stored procedure also resets the statistics. To show changes in database activity over time, three collection intervals are created.

In the first interval, run two scripts, `work1.db2` and `work2.db2`, then collect and reset the statistics.

```
db2 -o- -tvf work1.db2
db2 -o- -tvf work2.db2
CONNECT TO SAMPLE
CALL WLM_COLLECT_STATS
```

In the second interval, run only the `work1.db2` script once, then collect and reset the statistics.

```
db2 -o- -tvf work1.db2
CONNECT TO SAMPLE
CALL WLM_COLLECT_STATS
```

In the third interval, run the `work1.db2` script twice and the `work2.db2` script once, then collect and reset the statistics.

```
db2 -o- -tvf work1.db2
db2 -o- -tvf work2.db2
db2 -o- -tvf work1.db2
CONNECT TO SAMPLE
CALL WLM_COLLECT_STATS
```

Collecting data periodically such as this permits you to watch how work on your system changes over time.

Additional Information: Collecting data periodically does not need to be a manual operation. Using the WLM_COLLECT_INT database configuration parameter, you can set the interval in minutes after which statistics collection and reset occurs.
Step 5: Query views to view statistics

Now that statistics have been collected, the views created earlier can be used to look at the statistics. The HISTOGRAMTYPES view just returns the types of histograms available.

```sql
SELECT * FROM HISTOGRAMTYPES

<table>
<thead>
<tr>
<th>HISTOGRAM_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordActEstCost</td>
</tr>
<tr>
<td>CoordActExecTime</td>
</tr>
<tr>
<td>CoordActInterArrivalTime</td>
</tr>
<tr>
<td>CoordActLifeTime</td>
</tr>
<tr>
<td>CoordActQueueTime</td>
</tr>
</tbody>
</table>

3 record(s) selected.
```

Since the EXTENDED option was used when altering the service class, there are five histograms.

The HISTOGRAMSERVICECLASSES view permits you to see the service classes for which a histogram was collected. The following example restricts the output to that of the CoordActInterArrivalTime histogram only. Since aggregate activity collection was turned on only for the default user service class’s default subclass, only that class is shown when selecting from the HISTOGRAMSERVICECLASSES view.

```sql
SELECT * FROM HISTOGRAMSERVICECLASSES
WHERE HISTOGRAM_TYPE = 'CoordActInterArrivalTime'
ORDER BY SERVICE_SUPERCLASS, SERVICE_SUBCLASS

<table>
<thead>
<tr>
<th>HISTOGRAM_TYPE</th>
<th>SERVICE_SUPERCLASS</th>
<th>SERVICE_SUBCLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordActInterArrivalTime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
</tr>
</tbody>
</table>

1 record(s) selected.
```

The HISTOGRAMTIMES view shows the times when histograms were collected. Since the WLM_COLLECT_STATS procedure was run three times, there are three timestamps for the inter-arrival time histogram shown.

```sql
SELECT * FROM HISTOGRAMTIMES
WHERE HISTOGRAM_TYPE = 'CoordActInterArrivalTime'
AND SERVICE_SUPERCLASS = 'SYSDEFAULTUSERCLASS'
AND SERVICE_SUBCLASS = 'SYSDEFAULTSUBCLASS'
ORDER BY TIMESTAMP

<table>
<thead>
<tr>
<th>HISTOGRAM_TYPE</th>
<th>SERVICE_SUPERCLASS</th>
<th>SERVICE_SUBCLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordActInterArrivalTime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
</tr>
<tr>
<td>2007-08-08-13.41.38.870298</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoordActInterArrivalTime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
</tr>
<tr>
<td>2007-08-08-13.41.42.802855</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoordActInterArrivalTime</td>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
</tr>
<tr>
<td>2007-08-08-13.41.53.577835</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The last view, HISTOGRAMS, is for looking at the histograms themselves. Unlike the HISTOGRAMTIMES view that lists each collection interval as its own row, this view aggregates histogram data across multiple intervals to produce a single histogram of a given type for a given service class.
```
SELECT BIN_TOP, NUMBER_IN_BIN FROM HISTOGRAMS
WHERE HISTOGRAM_TYPE = 'CoordActInterArrivalTime'
  AND SERVICE_SUPERCLASS = 'SYSDEFAULTUSERCLASS'
  AND SERVICE_SUBCLASS = 'SYSDEFAULTSUBCLASS'
ORDER BY BIN_TOP

<table>
<thead>
<tr>
<th>BIN_TOP</th>
<th>NUMBER_IN_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>32</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>29</td>
<td>9</td>
</tr>
<tr>
<td>44</td>
<td>24</td>
</tr>
<tr>
<td>68</td>
<td>11</td>
</tr>
<tr>
<td>103</td>
<td>8</td>
</tr>
<tr>
<td>158</td>
<td>8</td>
</tr>
<tr>
<td>241</td>
<td>9</td>
</tr>
<tr>
<td>369</td>
<td>1</td>
</tr>
<tr>
<td>562</td>
<td>10</td>
</tr>
<tr>
<td>858</td>
<td>5</td>
</tr>
<tr>
<td>1309</td>
<td>5</td>
</tr>
<tr>
<td>1997</td>
<td>0</td>
</tr>
<tr>
<td>3046</td>
<td>0</td>
</tr>
<tr>
<td>4647</td>
<td>0</td>
</tr>
<tr>
<td>7089</td>
<td>0</td>
</tr>
<tr>
<td>10813</td>
<td>2</td>
</tr>
<tr>
<td>16493</td>
<td>2</td>
</tr>
<tr>
<td>25157</td>
<td>0</td>
</tr>
<tr>
<td>38373</td>
<td>0</td>
</tr>
<tr>
<td>58532</td>
<td>0</td>
</tr>
<tr>
<td>89280</td>
<td>0</td>
</tr>
<tr>
<td>136181</td>
<td>0</td>
</tr>
<tr>
<td>207720</td>
<td>0</td>
</tr>
<tr>
<td>316840</td>
<td>0</td>
</tr>
<tr>
<td>483283</td>
<td>0</td>
</tr>
<tr>
<td>737162</td>
<td>0</td>
</tr>
<tr>
<td>1124409</td>
<td>0</td>
</tr>
<tr>
<td>1715085</td>
<td>0</td>
</tr>
<tr>
<td>2616055</td>
<td>0</td>
</tr>
<tr>
<td>3990325</td>
<td>0</td>
</tr>
<tr>
<td>6086529</td>
<td>0</td>
</tr>
<tr>
<td>9283913</td>
<td>0</td>
</tr>
<tr>
<td>14160950</td>
<td>0</td>
</tr>
<tr>
<td>21600000</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Running this query produces output than will not be exactly the same as what is shown in the preceding example since activity inter-arrival times depend on the performance of the system. In the previous output, there are 41 bins and all of the largest bins are empty. At the top, there is a bin whose BIN_TOP is -1. This bin represents all of those activities whose inter-arrival time was too large to fit in the histogram. Seeing a NUMBER_OF_BIN greater than zero when the BIN_TOP is -1 indicates that you should probably increase the high bin value of your histogram. In the previous output, the NUMBER_IN_BIN is 0, so there is no need to make such a change. The majority of activities arrived less than 1309 milliseconds apart from each other. Four activities arrived between 7089 milliseconds and 16493 milliseconds apart.
The same query can be repeated with a histogram_type of CoordActEstCost instead of CoordActInterArrivalTime.

```sql
SELECT BIN_TOP, NUMBER_IN_BIN FROM HISTOGRAMS
WHERE HISTOGRAM_TYPE = 'CoordActEstCost'
  AND SERVICE_SUPERCLASS = 'SYSDEFAULTUSERCLASS'
  AND SERVICE_SUBCLASS = 'SYSDEFAULTSUBCLASS'
ORDER BY BIN_TOP
```

<table>
<thead>
<tr>
<th>BIN_TOP</th>
<th>NUMBER_IN_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>30</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>0</td>
</tr>
<tr>
<td>158</td>
<td>0</td>
</tr>
<tr>
<td>241</td>
<td>0</td>
</tr>
<tr>
<td>369</td>
<td>0</td>
</tr>
<tr>
<td>562</td>
<td>0</td>
</tr>
<tr>
<td>858</td>
<td>0</td>
</tr>
<tr>
<td>1309</td>
<td>0</td>
</tr>
<tr>
<td>1997</td>
<td>0</td>
</tr>
<tr>
<td>3046</td>
<td>0</td>
</tr>
<tr>
<td>4647</td>
<td>0</td>
</tr>
<tr>
<td>7089</td>
<td>0</td>
</tr>
<tr>
<td>10813</td>
<td>0</td>
</tr>
<tr>
<td>16493</td>
<td>0</td>
</tr>
<tr>
<td>25157</td>
<td>0</td>
</tr>
<tr>
<td>38373</td>
<td>0</td>
</tr>
<tr>
<td>58532</td>
<td>0</td>
</tr>
<tr>
<td>89280</td>
<td>0</td>
</tr>
<tr>
<td>136181</td>
<td>0</td>
</tr>
<tr>
<td>207720</td>
<td>0</td>
</tr>
<tr>
<td>316840</td>
<td>0</td>
</tr>
<tr>
<td>483283</td>
<td>0</td>
</tr>
<tr>
<td>737162</td>
<td>0</td>
</tr>
<tr>
<td>1124099</td>
<td>0</td>
</tr>
<tr>
<td>1715085</td>
<td>0</td>
</tr>
<tr>
<td>2616055</td>
<td>0</td>
</tr>
<tr>
<td>3990325</td>
<td>0</td>
</tr>
<tr>
<td>6086529</td>
<td>0</td>
</tr>
<tr>
<td>9283913</td>
<td>0</td>
</tr>
<tr>
<td>14160950</td>
<td>0</td>
</tr>
<tr>
<td>21600000</td>
<td>0</td>
</tr>
</tbody>
</table>

41 record(s) selected.

A histogram such as this is typical for a small workload. With a small workload, there is not much variation in the size of activities, so there are only three different bins that had activities counted towards them. Slightly more than 60% of the activities had a cost estimate between 5 and 19 timerons, with the rest having cost estimates of less than 1 timeron.
Step 6: Reset for other exercises

The final step is to turn off collection of aggregate activities on the default user service class, to drop the views and to delete the information in the statistics tables.

```
ALTER SERVICE CLASS SYSDEFAULTSUBCLASS UNDER SYSDEFAULTUSERCLASS
  COLLECT AGGREGATE ACTIVITY DATA NONE

DROP VIEW HISTOGRAMS
DROP VIEW HISTOGRAMTIMES
DROP VIEW HISTOGRAMSERVICECLASSES
DROP VIEW HISTOGRAMTYPES

SET EVENT MONITOR DB2STATISTICS STATE 0

DELETE FROM HISTOGRAMBIN DB2STATISTICS
DELETE FROM SCSTATS_DB2STATISTICS
```
Chapter 7. Workload management scenarios

Workload management sample application

Comprehensive workload management features have been integrated into your DB2 data server with DB2 workload management, giving you finer control over activities, resources and performance, and deeper insight into how your system is running. A workload management sample application is now available on developerWorks®.

The workload management sample application demonstrates how you can use DB2 workload management features to achieve the following objectives:

**Protect the system from runaway queries**
Runaway queries are costly and cause poor performance. The workload management sample application identifies queries with the potential to become runaway queries, and then stops these queries from running after they have violated a specified threshold.

**Limit concurrent resource consumption by individual applications**
The sample application shows how to use DB2 workload management features to prevent applications that submit large amounts of concurrent work from negatively affecting the performance of other applications.

**Achieve a specific response time**
Workload management features permit you to achieve a specific response time objective of the form: “transaction X from application Y shall complete within 1 second in 90% of cases,” regardless of what other activity is running concurrently on the system. The sample application will demonstrate how to achieve a response time objective.

**Consistent response time for short queries**
Queries that typically have a response time of less than 1 second should have a relatively consistent response time regardless of what other workloads are running on the system. The sample application uses the query execution time histogram to monitor consistency.

**Protect the system during periods of peak demand**
Workload management policy features protect the system from capacity overload during bursts of peak demand by queuing work once the system is sufficiently loaded.

**Enable concurrent batch extract, transform, and load (ETL) processing and user queries**
Workload management features permit you to run ETL jobs (like loading data into tables) while controlling the performance impact for users running queries concurrently.

To obtain the sample application, see Workload management sample on developerWorks.
Scenario: Investigating a workload-related system slowdown

If you notice a system slowdown (for example, some applications take much longer to complete than expected) and are unsure whether the problem is related to the configuration of the workloads, you can use table function data to investigate and, if necessary, correct the problem.

First, create a query that aggregates data across service classes and database members using data from the WLM_GET_SERVICE_SUBCLASS_STATS table function. Set the first and second arguments to empty strings and the third argument to -2 to indicate that data is to be gathered for all service classes on all database members.

Your query might resemble the following one:

```sql
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(SUM(COORD_ACT_COMPLETED_TOTAL)),1,13) AS ACTSCOMPLETED,
       SUBSTR(CHAR(SUM(COORD_ACT_ABORTED_TOTAL)),1,11) AS ACTSABORTED,
       SUBSTR(CHAR(MAX(CONCURRENT_ACT_TOP)),1,6) AS ACTSHW,
       CAST(CASE WHEN SUM(COORD_ACT_COMPLETED_TOTAL) = 0 THEN 0
                  ELSE SUM(COORD_ACT_COMPLETED_TOTAL * COORD_ACT_LIFETIME_AVG)
                       / SUM(COORD_ACT_COMPLETED_TOTAL) END / 1000 AS DECIMAL(9,3))
          AS ACTAVGLIFETIME
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '', -2)) AS SCSTATS
GROUP BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME
```

In the preceding example data, the SUB1 service subclass in the SUP1 service superclass is running more simultaneous activities than usual. To investigate further, you might want to examine the statistics for workloads that map to this service class. Your query might resemble the following one:

```sql
SELECT SUBSTR(WLSTATS.WORKLOAD_NAME,1,22) AS WL_NAME,
       SUBSTR(WLSTATS.MEMBER),4) AS MEMB,
       CONCURRENT_WLO_TOP AS WLO_HIGH_WTRMRK,
       CONCURRENT_WLO_ACT_TOP AS WLO_ACT_HIGH_WTRMRK
FROM TABLE(WLM_GET_WORKLOAD_STATS('', -2)) AS WLSTATS,
     TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', -2)) AS SCWLOS
WHERE WLSTATS.WORKLOAD_NAME = SCWLOS.WORKLOAD_NAME
AND SCWLOS.SERVICE_SUPERCLASS_NAME = 'SUP1'
AND SCWLOS.SERVICE_SUBCLASS_NAME = 'SUB1'
ORDER BY WL_NAME, MEMB;
```

The output shows that an application in the LYNNSALES workload submitted 8 activities concurrently. Consider adding a threshold to restrict concurrency of coordinator activities for each workload occurrence.
Scenario: Aggregation of activity metrics across child activities

Each row in an activity event monitor contains information and monitor metrics describing the execution of a specific activity. Included in the activity information are the `parent_uow_id` and `parent_activity_id` monitor elements. These monitor elements are nonzero if the activity is a child of some other activity (nested) and the elements identify the parent activity.

Using recursive SQL, it is possible to aggregate the monitor metrics in the activity event monitor across all child activities to their parent activity. Viewing the aggregated metrics for the parent is useful in identifying which parts of a stored procedure may benefit from tuning. For example, if a stored procedure P1 calls stored procedures P2 and P3, the aggregated metrics might show that 90% of the CPU usage during a call to P1 occurs during the processing of statements in the nested procedure P3. As a result, you can focus your investigation accordingly.

You can use the following as an example of how to obtain aggregates of nested metrics from an activity event monitor.

**Note:** The `@` character is used as a statement termination character in this example. The following statements can be copied to a file (for example, `test.clp`) and executed using `db2 -td@ -f filename`, as shown in the following example command:

```
db2 -td@ -f test.clp
```

For the purpose of this example, suppose the following tables and stored procedures exist:

```sql
CREATE TABLE T1 (ONE INT)
DROP PROCEDURE TEST.P10
DROP PROCEDURE TEST.P20
DROP PROCEDURE TEST.P30
drop procedure TEST.P40

CREATE PROCEDURE TEST.P4()
LANGUAGE SQL
BEGIN
  INSERT INTO T1 VALUES(5);
  INSERT INTO T1 VALUES(6);
  INSERT INTO T1 VALUES(7);
END

CREATE PROCEDURE TEST.P3()
LANGUAGE SQL
BEGIN
  DECLARE V INTEGER;
  INSERT INTO T1 VALUES(1);
  CALL TEST.P4();
  SELECT COUNT(*) INTO V FROM T1;
END

CREATE PROCEDURE TEST.P2()
LANGUAGE SQL
BEGIN
  INSERT INTO T1 VALUES(2);
  INSERT INTO T1 VALUES(3);
END

CREATE PROCEDURE TEST.P1()
LANGUAGE SQL
```
BEGIN
CALL TEST.P3();
CALL TEST.P2();
INSERT INTO T1 VALUES(4);
END

**Procedure**

1. Create an activity event monitor and enable activity capture. In this example, activity capture is enabled for the current connection using the WLM_SET_CONN_ENV procedure. Execute the TEST.P1 procedure using the same connection and then disable the activity event monitor. Activity information will be captured for the TEST.P1 procedure and all of its child activities.

   ```sql
   CREATE EVENT MONITOR A FOR ACTIVITIES WRITE TO TABLE
   SET EVENT MONITOR A STATE 1
   CALL WLM_SET_CONN_ENV(NULL, '<collectactdata>WITH DETAILS</collectactdata>
<collectactpartition>ALL</collectactpartition>')</call>
   CALL TEST.P1()
   SET EVENT MONITOR A STATE 0
   CALL WLM_SET_CONN_ENV(NULL, '<collectactdata>NONE</collectactdata>')
   ```

2. Execute the following query to display the captured statements, along with their parent identification information, individual CPU consumption, and aggregate CPU consumption. The aggregate CPU consumption is recursively summed across all child activities. This query can be easily extended to support any of the monitoring metrics captured by the activity event monitor.

   ```sql
   WITH ACT( APPL_ID,
   UOW_ID,
   ACTIVITY_ID,
   PARENT_UOW_ID,
   PARENT_ACTIVITY_ID,
   CPU )
AS ( SELECT APPL_ID,
   UOW_ID,
   ACTIVITY_ID,
   MAX(PARENT_UOW_ID),
   MAX(PARENT_ACTIVITY_ID),
   SUM(METRICS.TOTAL_CPU_TIME)
   FROM ACTIVITY_A AS A,
   XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
   'actmetrics/activity_metrics' PASSING XMLPARSE(DOCUMENT A.DETAILS_XML) as "actmetrics"
   COLUMNS
   TOTAL_CPU_TIME BIGINT PATH 'total_cpu_time' ) AS METRICS
   WHERE A.PARTIAL_RECORD = 0
   GROUP BY APPL_ID,
   UOW_ID,
   ACTIVITY_ID ),
TMP( BASE_APPL_ID,
BASE_UOW_ID,
BASE_ACTIVITY_ID,
APPL_ID,
UOW_ID,
ACTIVITY_ID,
PARENT_UOW_ID,
PARENT_ACTIVITY_ID,
CPU,
LEVEL )
AS ( SELECT APPL_ID,
   UOW_ID,
   ACTIVITY_ID,
   PARENT_UOW_ID,
PARENT_ACTIVITY_ID,
   CPU,
   LEVEL )
   ```

*Note:* To improve readability, only CPU consumption is reported here.
UOW_ID,
ACTIVITY_ID,
APPL_ID,
UOW_ID,
ACTIVITY_ID,
PARENT_UOW_ID,
ACTIVITY_ID,
PARENT_ACTIVITY_ID,
CPU,
1
FROM ACT
UNION ALL
SELECT T.BASE_APPL_ID,
T.BASE_UOW_ID,
T.BASE_ACTIVITY_ID,
A.APPL_ID,
A.UOW_ID,
A.ACTIVITY_ID,
A.PARENT_UOW_ID,
A.PARENT_ACTIVITY_ID,
A.CPU,
T.LEVEL + 1
FROM ACT AS A, TMP AS T
WHERE A.APPL_ID = T.APPL_ID AND
A.PARENT_UOW_ID = T.UOW_ID AND
A.PARENT_ACTIVITY_ID = T.ACTIVITY_ID AND
T.LEVEL < 128 ),
AGG( APPL_ID,
UOW_ID,
ACTIVITY_ID,
CPU )
AS ( SELECT BASE_APPL_ID,
BASE_UOW_ID,
BASE_ACTIVITY_ID,
SUM(CPU)
FROM TMP
GROUP BY BASE_APPL_ID,
BASE_UOW_ID,
BASE_ACTIVITY_ID )
SELECT
A.UOW_ID,
A.ACTIVITY_ID,
A.PARENT_UOW_ID,
A.PARENT_ACTIVITY_ID,
A.CPU AS STMT_CPU,
B.CPU AS AGG_CPU,
SUBSTR(CONCAT(REPEAT(' ',C.STMT_NEST_LEVEL),
C.STMT_TEXT),
1, 30) AS STMT_TEXT
FROM ACT AS A,
AGG AS B,
ACTIVITY_STMT_A AS C,
ACTIVITY_A AS D
WHERE A.APPL_ID = B.APPL_ID AND
A.UOW_ID = B.UOW_ID AND
A.ACTIVITY_ID = B.ACTIVITY_ID AND
D.COORD_PARTITION_NUM = D.PARTITION_NUMBER AND
A.APPL_ID = C.APPL_ID AND
A.UOW_ID = C.UOW_ID AND
A.ACTIVITY_ID = C.ACTIVITY_ID AND
A.APPL_ID = D.APPL_ID AND
A.UOW_ID = D.UOW_ID AND
A.ACTIVITY_ID = D.ACTIVITY_ID AND
D.PARTIAL_RECORD = 0
ORDER BY D.TIME_CREATED ASC@
This query produces the following output. The STMT_CPU column reports the CPU consumption for the statement without including CPU usage of any child activities. The AGG_CPU column reports the aggregate CPU consumption for the activity and all of its children.

**Note:** If queries execute too quickly to be measurable, STMT_CPU and AGG_CPU might be zero.

<table>
<thead>
<tr>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>PARENT_UOW_ID</th>
<th>PARENT_ACTIVITY_ID</th>
<th>STMT_CPU</th>
<th>AGG_CPU</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>576</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5353</td>
<td>84064</td>
<td>CALL TEST.P1()</td>
</tr>
<tr>
<td>576</td>
<td>2</td>
<td>576</td>
<td>1</td>
<td>7444</td>
<td>52043</td>
<td>CALL TEST.P2()</td>
</tr>
<tr>
<td>576</td>
<td>3</td>
<td>576</td>
<td>2</td>
<td>1869</td>
<td>1869</td>
<td>INSERT INTO T1 VALUES(1)</td>
</tr>
<tr>
<td>576</td>
<td>4</td>
<td>576</td>
<td>2</td>
<td>11727</td>
<td>26935</td>
<td>CALL TEST.P3()</td>
</tr>
<tr>
<td>576</td>
<td>5</td>
<td>576</td>
<td>4</td>
<td>2017</td>
<td>2017</td>
<td>INSERT INTO T1 VALUES(5)</td>
</tr>
<tr>
<td>576</td>
<td>6</td>
<td>576</td>
<td>4</td>
<td>6602</td>
<td>6602</td>
<td>INSERT INTO T1 VALUES(6)</td>
</tr>
<tr>
<td>576</td>
<td>7</td>
<td>576</td>
<td>4</td>
<td>6589</td>
<td>6589</td>
<td>INSERT INTO T1 VALUES(7)</td>
</tr>
<tr>
<td>576</td>
<td>8</td>
<td>576</td>
<td>2</td>
<td>15795</td>
<td>15795</td>
<td>SELECT COUNT(*) INTO :HV00</td>
</tr>
<tr>
<td>576</td>
<td>9</td>
<td>576</td>
<td>1</td>
<td>11727</td>
<td>20314</td>
<td>CALL TEST.P2()</td>
</tr>
<tr>
<td>576</td>
<td>10</td>
<td>576</td>
<td>9</td>
<td>1941</td>
<td>1941</td>
<td>INSERT INTO T1 VALUES(2)</td>
</tr>
<tr>
<td>576</td>
<td>11</td>
<td>576</td>
<td>9</td>
<td>6646</td>
<td>6646</td>
<td>INSERT INTO T1 VALUES(3)</td>
</tr>
<tr>
<td>576</td>
<td>12</td>
<td>576</td>
<td>1</td>
<td>6354</td>
<td>6354</td>
<td>INSERT INTO T1 VALUES(4)</td>
</tr>
</tbody>
</table>

12 record(s) selected.

**Scenario: Identifying activities that are taking too long to complete**

Workload management table functions simplify the task of identifying a specific activity inside the data server and, if necessary, canceling it without having to end the entire application.

**Identifying an activity that is taking too long to complete**

Following is an example of identifying a long-running query. Assume that a user from the Sales department who is running the SalesReport application complains that the application is taking too long to complete.

After identifying the application handle, use the `WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES` table function to look up all activities currently running in this application. For example, if the application handle is 1, your query might resemble the following one:

```sql
SELECT SUBSTR(CHAR(COORD_MEMBER),1,5) AS COORD,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(UOW_ID),1,5) AS UOWID,
       SUBSTR(CHAR(ACTIVITY_ID),1,5) AS ACTID,
       SUBSTR(CHAR(PARENT_UOW_ID),1,8) AS PARUOWID,
       SUBSTR(CHAR(PARENT_ACTIVITY_ID),1,8) AS PARACTID,
       SUBSTR(ACTIVITY_TYPE,1,8) AS ACTTYPE,
       SUBSTR(CHAR(NESTING_LEVEL),1,7) AS NESTING
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2))
AS WLOACTS
ORDER BY MEMB, UOWID, ACTID
```

| COORD MEMB UOWID ACTID PARUOWID PARACTID ACTTYPE NESTING |
|------------|---------|--------|----------|--------|
| 0          | 0       | 2      | 3        | -      | CALL 0 |
| 0          | 0       | 2      | 5        | 2      | 3 READ_DML 1 |

The activity is identified as having a unit of work ID of 2 and an activity ID of 5. You can then use the `WLM_GET_SERVICE_CLASS_AGENTS` table function to discover what the agents that work on this activity are doing:

```sql
SELECT APPLICATION_HANDLE, UOW_ID, ACTIVITY_ID,
       SUBSTR(REQUEST_TYPE,1,8) AS REQUEST_TYPE,
       SUBSTR(EVENT_TYPE,1,8) AS EVENT_TYPE,
       SUBSTR(EVENT_OBJECT,1,8) AS EVENT_OBJECT
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2))
AS WLOACTS
```
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS('', '', CAST(NULL AS BIGINT),-2))
AS AGENTS
WHERE APPLICATION_HANDLE = 1
AND UOW_ID = 2
AND ACTIVITY_ID = 5

For example, the activity might be queued, executing, or waiting on a lock. If the activity were queued, the result would be:

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>REQUEST_TYPE</th>
<th>EVENT_TYPE</th>
<th>EVENT_OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>OPEN</td>
<td>WAIT</td>
<td>WLM_QUEUE</td>
</tr>
</tbody>
</table>

If the activity were executing, the result would be:

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>REQUEST_TYPE</th>
<th>EVENT_TYPE</th>
<th>EVENT_OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>OPEN</td>
<td>PROCESS</td>
<td>REQUEST</td>
</tr>
</tbody>
</table>

If the activity were waiting on a lock, the result would be:

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>REQUEST_TYPE</th>
<th>EVENT_TYPE</th>
<th>EVENT_OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>OPEN</td>
<td>ACQUIRE</td>
<td>LOCK</td>
</tr>
</tbody>
</table>

When you know what the activity is doing, you can proceed appropriately:

- If the activity is queued, if the user indicates that the query is taking so long that they no longer care about the results, or you think the query is consuming too many resources, you can cancel it.
- If the activity is important and it is queued, consider cancelling some other less important work that is currently running (reducing the concurrency so that activities leave queue), or maybe the user will be satisfied to know that work is not hanging and is just waiting for chance to run.
- If the activity is waiting for a lock, you can use the snapshot monitor to investigate which locks the application is waiting for.
- If the activity is waiting for a lock held by lower priority activity, consider cancelling that activity.

You might also find it useful to know the DML statement that activity 5 is running. Assuming that you have an active activities event monitor, you can run the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure to capture information about the DML statement and other information about activity 5 while it is running. Unlike the statement event monitor, the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure permits you to capture information about a specific query, as opposed to every statement running at the time. You can also obtain the statement text by using MON_GET_ACTIVITY_DETAILS.

If you decide that you must cancel the activity, you can use the WLM_CANCEL_ACTIVITY routine to cancel the activity without having to end the application that issued it:

```
CALL WLM_CANCEL_ACTIVITY (1, 2, 5)
```

The application that issued the activity receives an SQL4725N error. Any application that handles negative SQL codes is able to handle this SQL code.
Identifying an activity hang caused by lock contention

Assume that you have a situation in which a user is complaining about an application that is taking too long. Also assume that you have either the application name or the authorization ID of the long-running application. With this information, you can use the `LIST APPLICATIONS` command to obtain the application handle. Assuming that application handle returned by the `LIST APPLICATIONS` command is 2, you can use the `WLM_GET_SERVICE_CLASS_AGENTS` table function to determine which agents are working on this activity. Your query might resemble the following one:

```sql
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
       SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
       SUBSTR(EVENT_OBJECT,1,11) AS EVENTOBJECT,
       SUBSTR(REQUEST_TYPE,1,7) AS REQTYPE,
       SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
       SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_AGENTS('', '', 2, -2)) AS SCDETAILS
ORDER BY APPHANDLE, MEMB, AGENT_TID
```

<table>
<thead>
<tr>
<th>APPHANDLE</th>
<th>MEMB</th>
<th>AGENT_TID</th>
<th>AGENTTYPE</th>
<th>EVENTOBJECT</th>
<th>REQTYPE</th>
<th>UOW_ID</th>
<th>ACT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>COORDINATOR</td>
<td>REQUEST</td>
<td>OPEN</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>SUBAGENT</td>
<td>LOCK</td>
<td>-</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

The results indicate that agent 1 is waiting on a remote response. Looking at the agent on the remote member that is working on the same activity, the `EVENTOBJECT` field indicates that the agent is waiting to obtain a lock.

The next step is to determine who owns the lock. You can obtain this information by turning on the monitor switches and using the snapshot monitor table function, as shown in the following example:

```sql
SELECT WAITING_FOR_LOCK AS WAITING_FOR_LOCK,
       HOLDING_LOCK AS HOLDING_LOCK,
       CAST(LOCK_MODE_REQUESTED AS SMALLINT) AS WANTED,
       CAST(LOCK_MODE AS SMALLINT) AS HELD
FROM TABLE(MON_GET_APPL_LOCKWAIT (null, -1)) AS SLW
```

<table>
<thead>
<tr>
<th>WAITING_FOR_LOCK</th>
<th>HOLDING_LOCK</th>
<th>WANTED</th>
<th>HELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

You can also determine the lock owner by using the following sequence of commands:

```
db2pd -db database alias -locks
db2pd -db database alias -transactions
```

If you want to cancel the long-running activity, you can use the `WLM_CANCEL_ACTIVITY` procedure. If the successful completion of the long-running application is more important than the successful completion of the lock-owning application, you can force the lock-owning application.

---

**Scenario: How to cancel activities queued for more than an hour**

Using the example scripts described here, you can create a procedure to cancel activities that have been queued for more than an hour. In addition, an example script is provided that can be used to schedule the queued-activity-cancelling procedure to run every 10 minutes using the DB2 Administrative Task Scheduler.
The queued-activity-cancelling procedure also captures information about the cancelled activities (if an activity event monitor is active), and maintains a small history table of cancelled activities. Both of these informational components are optional and comments in the example script indicate where to comment out the components, if they are not required.

The statements contained in the example procedure are themselves activities and subject to threshold control (depending on how thresholds are configured on your system). Consider running the example queued-activity-cancelling procedure in a service class that does not have any queuing thresholds applied.

1. Copy the following example script, that creates the procedure to cancel activities queued for more than 1 hour, into a file you have created (for example, a file named Sample.x.clp):

```sql
-- Simple history table to track cancelled activities
CREATE TABLE SAMPLE.CANCELED_ACTIVITIES(
    APPLICATION_HANDLE BIGINT,
    UOW_ID BIGINT,
    ACTIVITY_ID BIGINT )

-- Cancel any activities that have been queued for more than 1 hour
CREATE PROCEDURE SAMPLE.CANCEL_QUEUED_ACTIVITIES()
LANGUAGE SQL
BEGIN
DECLARE APPHANDLE BIGINT;
DECLARE UOWID BIGINT;
DECLARE ACTIVITYID BIGINT;
DECLARE QUEUETIME BIGINT;
DECLARE AT_END INT DEFAULT 0;
DECLARE QUEUEDAPPS CURSOR WITH HOLD FOR SELECT APPLICATION_HANDLE,
    UOW_ID, ACTIVITY_ID
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(NULL,-2)) AS T
WHERE ACTIVITY_STATE = 'QUEUED' AND LOCAL_START_TIME IS NULL;
DECLARE QTIMECUR CURSOR FOR SELECT TIMESTAMPDIFF(8, CHAR
    (CURRENT_TIMESTAMP - TIMESTAMP(VALUE)))
FROM TABLE(WLM_GET_ACTIVITY_DETAILS(APPHANDLE ,
    UOWID , ACTIVITYID , -2)) AS T WHERE NAME = 'ENTRY_TIME';
DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET AT_END = 1;
-- Ignore errors for activity not found and activity event monitor does not exist.
DECLARE CONTINUE HANDLER FOR SQLSTATE '5U035', SQLSTATE '01H53'
    BEGIN
    END;
-- Find all activities that are queued by WLM
-- thresholds where (ACTIVITY_STATE = 'QUEUED')
OPEN QUEUEDAPPS;
FETCH QUEUEDAPPS INTO APPHANDLE, UOWID, ACTIVITYID;
WHILE AT_END = 0 DO
-- Now use activity entry time to estimate the time spent queued.
-- Queuing occurs before an activity begins execution, so queue
-- time is approximated using current time - entry time
OPEN QTIMECUR;
FETCH QTIMECUR INTO QUEUETIME;
```

Chapter 7. Workload management scenarios 355
CLOSE QTIMECUR;

IF ( QUEUETIME >= 1 ) THEN

-- Optional: Insert a record into a table to record the
-- cancellation of the statement (for monitoring purposes, to
-- understand how many statements were cancelled). Modify this
-- insert as required to capture more info such as the name of
-- the application that submitted the cancelled query. Comment out
-- these 2 lines if the monitoring is not important to you.
INSERT INTO SAMPLE.CANCELED_ACTIVITIES VALUES ( APPHANDLE,
                                                UOWID, ACTIVITYID );

-- Optional: Send details about activity to any activity activities
-- event monitor before cancelling. Comment out
-- this line if you don't care about the details of the
-- statements that were cancelled
CALL WLM_CAPTURE_ACTIVITY_IN_PROGRESS( APPHANDLE, UOWID,
                                       ACTIVITYID );

-- Cancel the activity
CALL WLM_CANCEL_ACTIVITY( APPHANDLE, UOWID, ACTIVITYID );

-- Explicit commit, required for the insert statement above. The
-- admin task scheduler will not perform a commit. Comment out this
-- line if the insert statement is removed.
COMMIT;

END IF;

FETCH QUEUEDAPPS INTO APPHANDLE, UOWID, ACTIVITYID;

END WHILE;

CLOSE QUEUEDAPPS;

END}

2. Create the queued-activity-cancelling procedure by executing script x.clp using
the following command:
   db2 -td0 -f x.clp

3. Execute the queued-activity-cancelling procedure by issuing the following
command:
   db2 "call sample.cancel_queued_activities()"
   Any activities that have been queued for more than 1 hour will be cancelled.

4. The following example script schedules the queued-activity-cancelling
   procedure to run every 10 minutes using the DB2 Administrative Task
   Scheduler. Copy the example script into a file you have created (for example, a
   file named y.clp):
   
   ---------------------------------------
   -- Enable DB2 Admin Task Scheduler if
   -- not already enabled.
   ---------------------------------------

   !db2set DB2_ATS_ENABLE=YES

   ---------------------------------------
   -- Create SYSTOOLSPACE tablespace.
   -- Enable if SYSTOOLSPACE does not already
   -- exist on your database.
   ---------------------------------------

   -- CREATE TABLESPACE SYSTOOLSPACE IN IBMCATGROUP MANAGED BY AUTOMATIC STORAGE
   -- EXTENTSIZE 40
-- Add a task to automatically cancel activities that have been queued for more than 1 hour. Task is scheduled to run every 10 minutes. Adjust the schedule as necessary using the schedule input parameter (specified in cron format).

CALL SYSPROC.ADMIN_TASK_ADD('CANCEL ACTIVITIES QUEUED FOR MORE 1 HOUR', NULL, NULL, NULL, '*/10 *** ***', 'SAMPLE', 'CANCEL_QUEUED_ACTIVITIES', NULL, NULL, NULL)@5

5. Schedule the queued-activity-cancelling procedure to run every 10 minutes by executing script y.clp using the following command:
   db2 -td@ -f y.clp

Scenario: Identifying activities with low estimated cost and high runtime

The following example shows how you can use work classes, work action sets, thresholds, and activity collection to identify activities that have a low estimated cost but a high runtime. This situation could indicate that the estimated cost (in timerons) is inaccurate because of out-of-date table and index statistics.

The first step is to create a work class set with a work class that will be used to identify activities with a low estimated cost. For example:

CREATE WORK CLASS SET WCS1
(WORK CLASS SMALLDML WORK TYPE DML FOR TIMERONCOST FROM 0 TO 500)

Then, you would create a database work action set with a work action that applies an activity-total-time threshold to the SMALLDML work class. The threshold action is CONTINUE and the COLLECT ACTIVITY DATA option is specified so that an activity that violates the threshold is sent to the activities event monitor on completion:

CREATE WORK ACTION SET WAS1 FOR DATABASE USING WORK CLASS SET WCS1
(WORK ACTION WA1 ON WORK CLASS SMALLDML WHEN ACTIVITYTOTALTIME > 15 MINUTES COLLECT ACTIVITY DATA WITH DETAILS CONTINUE)

Finally, you would create and activate a threshold violations event monitor and an activities event monitor:

CREATE EVENT MONITOR THVIOLATIONS FOR THRESHOLD VIOLATIONS WRITE TO TABLE SET EVENT MONITOR THVIOLATIONS STATE 1

CREATE EVENT MONITOR DB2ACTIVITIES FOR ACTIVITIES WRITE TO TABLE SET EVENT MONITOR DB2ACTIVITIES STATE 1

Now when a DML activity with an estimated cost of less than 500 timerons runs for greater than 15 minutes, a threshold violation record is written to the THVIOLATIONS event monitor (indicating that the total time threshold was
violated), and details about the DML activity are collected when the activity completes and sent to the DB2ACTIVITIES event monitor. You can use the information collected about the activity in the DB2ACTIVITIES event monitor to investigate further. For example, you can run the EXPLAIN statement on the query and examine the access plan. You should also consider the system load and queuing at the time the activity was collected, as a long lifetime can be a result of insufficient system resources or the activity being queued. The long lifetime does not necessarily indicate out-of-date statistics.

Scenario: Cancelling all activities being executed in a service subclass

The following is an example of a stored procedure that can be used to cancel all currently executing activities in a particular service subclass.

To create the CANCELALL procedure, execute the following steps:

1. Copy the following CREATE PROCEDURE statement to a file (for example, cancelall.ddl):

   ```sql
   CREATE PROCEDURE CANCELALL ( IN INSCID BIGINT )
   SPECIFIC CANCELALL
   LANGUAGE SQL
   BEGIN
   DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
   DECLARE APPHNDL BIGINT;
   DECLARE UOWID INTEGER;
   DECLARE ACTIVITYID INTEGER;
   DECLARE C1 CURSOR FOR (SELECT APPLICATION_HANDLE,
   UOW_ID, ACTIVITY_ID
   FROM TABLE(SYSPROC.WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES
   ( NULL, -2 ))
   AS T WHERE T.SERVICE_CLASS_ID = INSCID);
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
   BEGIN
   END;
   OPEN C1;
   FETCH_LOOP:
   LOOP
   FETCH C1 INTO APPHNDL, UOWID, ACTIVITYID;
   IF (SQLSTATE <> '00000')
   THEN LEAVE FETCH_LOOP;
   END IF;
   CALL WLM_CANCEL_ACTIVITY( APPHNDL, UOWID, ACTIVITYID );
   END LOOP FETCH_LOOP;
   END
   ```

2. Run the following CLP command:

   ```bash
   db2 -td@ -f cancelall.ddl
   ```

   After the procedure has been created, execute the procedure (for example, cancelling all activities in the service subclass which has ID = 15) using the following statement:

   ```sql
   CALL CANCELALL( 15 )
   ```
Note: The CANCELALL procedure must be run in a different service subclass from the target that is passed as input, otherwise the procedure will cancel itself.

Scenario: Disconnecting all applications either mapped to or executing activities in a service class

The following is an example of a stored procedure that can be used to disconnect (force) all applications that are either mapped to a particular service class or currently executing activities in a particular service class.

To create the FORCEALLINSC procedure, execute the following steps:

1. Copy the following CREATE PROCEDURE statement to a file (for example, forceall.ddl):

```sql
CREATE PROCEDURE FORCEALLINSC ( IN INSCID BIGINT )
    SPECIFIC FORCEALLINSC
    LANGUAGE SQL
BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE APPHNDL BIGINT;
    DECLARE UOWID INTEGER;
    DECLARE ACTIVITYID INTEGER;

    DECLARE C1 CURSOR FOR (SELECT APPLICATION_HANDLE
        FROM TABLE(SYSPROC.WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES
            ( NULL, -2 )
        ) AS T WHERE T.SERVICE_CLASS_ID = INSCID);

    DECLARE C2 CURSOR FOR (SELECT APPLICATION_HANDLE
        FROM TABLE(SYSPROC.WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES
            (NULL, NULL, -2 ))
        AS T, SYSCAT.SERVICECLASSES AS S
        WHERE T.SERVICE_SUPERCLASS_NAME = S.PARENTSERVICECLASSNAME AND
        T.SERVICE_SUBCLASS_NAME = S.SERVICECLASSNAME AND
        S.SERVICECLASSID = INSCID);

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        BEGIN
        END;

    -- First force any applications that have an activity running in
    -- the specified service class

    OPEN C1;
    FETCH_LOOP:
        LOOP
            FETCH C1 INTO APPHNDL;
            IF (SQLSTATE <> '00000')
                THEN LEAVE FETCH_LOOP;
            END IF;
            CALL ADMIN_CMD( 'FORCE APPLICATION (' || CHAR(APPHNDL) || ')' );
        END LOOP FETCH_LOOP;

    -- Now force any connections that are mapped to the service class, but which
    -- don't currently have any activities running

    OPEN C2;
```
Scenario: Tuning a DB2 workload management configuration when capacity planning data is available

If you performed capacity planning, you should have information about the types of users and their expected response times. You can use this information to construct, determine the effectiveness of, and tune your DB2 workload management configuration.

Assume that you performed capacity planning and that the data in the following table represents the results of this exercise for work types and response time goals:

<table>
<thead>
<tr>
<th>Type of work</th>
<th>Application</th>
<th>Goal</th>
<th>Importance</th>
<th>Expected throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order entry</td>
<td>orderentryapp.exe</td>
<td>Obtain an average response time &lt; 1 second</td>
<td>High</td>
<td>10 000 (both inserts and updates) per day</td>
</tr>
<tr>
<td>Business intelligence queries</td>
<td>businessobjects.exe</td>
<td>Obtain an average response time &lt; 10 seconds</td>
<td>High</td>
<td>100 queries per day</td>
</tr>
<tr>
<td>Batch processing</td>
<td>batchapp.exe</td>
<td>Maximize throughput</td>
<td>Low</td>
<td>5000 updates per day</td>
</tr>
<tr>
<td>Other</td>
<td>All other applications</td>
<td>Best effort</td>
<td>Low</td>
<td>100 activities per day</td>
</tr>
</tbody>
</table>

Based on the data in the preceding table, you might create three service classes (ORDER_ENTRY_SC, BI_QUERIES_SC, and BATCH_SC) and three workloads (ORDER_ENTRY_WL, BI_QUERIES_WL, and BATCH_WL) to assign work to the service classes. After creating the service classes and workloads, you might create a statistics event monitor to collect aggregate activity information, such as the activity lifetime histogram for each service class. Assume that the data in the following table compares the average daily count of activities in each service class.
(computed from the activity lifetime histogram) with the volumes that were predicted in the capacity planning exercise:

<table>
<thead>
<tr>
<th>Service class</th>
<th>Predicted activities per day</th>
<th>Actual activities per day</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER_ENTRY_SC</td>
<td>10 000</td>
<td>9700</td>
</tr>
<tr>
<td>BI_QUERIES_SC</td>
<td>100</td>
<td>115</td>
</tr>
<tr>
<td>BATCH_SC</td>
<td>5000</td>
<td>5412</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>100</td>
<td>85</td>
</tr>
</tbody>
</table>

The observed data indicates that the capacity planning estimates were accurate. The data in the following table compares the average activity lifetimes (obtained from the activity lifetime histogram) with the response time goals determined during capacity planning and shows that activities in the BI_QUERIES_SC service class are not meeting their response time objectives.

<table>
<thead>
<tr>
<th>Service class</th>
<th>Response time goal</th>
<th>Actual average lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER_ENTRY_SC</td>
<td>&lt; 1 second</td>
<td>0.8 seconds</td>
</tr>
<tr>
<td>BI_QUERIES_SC</td>
<td>&lt; 10 seconds</td>
<td>30 seconds</td>
</tr>
<tr>
<td>BATCH_SC</td>
<td></td>
<td>2 seconds</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td></td>
<td>10 minutes</td>
</tr>
</tbody>
</table>

With DB2 workload management, you can use different approaches when addressing the problem of the business intelligence queries not meeting their response time goals:

- Limiting the concurrency of lower-importance service classes
- Allowing the operating system workload manager to provide less processor resource to less-important service classes
- Modifying the agent and I/O prefetcher priorities for the service classes
- Using any combination of the previous three approaches

Assume that processor time is the resource that is causing the business intelligence queries to fail to meet their goals. Also assume that you use the operating system workload manager to give the SYSDEFAULTUSERCLASS service class less processor resources than other service classes. You can then capture aggregate activity information over a period of days to observe whether the changes to the CPU allocation provide the results that you expect. The data in the following table shows another comparison between response time goals and actual average lifetimes computed from the histograms after you made the operating system workload manager changes. All service classes are now meeting their response time objectives and, because of the processor time reallocation, activities in the SYSDEFAULTUSERCLASS service class have had their response times doubled.

<table>
<thead>
<tr>
<th>Service class</th>
<th>Response time goal</th>
<th>Actual average lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER_ENTRY_SC</td>
<td>&lt; 1 second</td>
<td>0.6 seconds</td>
</tr>
<tr>
<td>BI_QUERIES_SC</td>
<td>&lt; 10 seconds</td>
<td>9.5 seconds</td>
</tr>
<tr>
<td>BATCH_SC</td>
<td></td>
<td>1.5 seconds</td>
</tr>
</tbody>
</table>
Scenario: Tuning a DB2 workload management configuration when capacity planning information is unavailable

You can use the DB2 workload management tools to help design, monitor, and tune a workload management configuration even if you do not have capacity analysis data to use for designing the configuration.

Assume that you do not initially know which workloads and service classes to create because either you do not have full knowledge of the workload on the system or you do not yet know which workloads are required for stable execution results. Also assume that you know that some applications have response time requirements but that you do not yet know how many other applications are competing for resources with such time-critical applications. You can use the workload management monitoring capabilities to determine this.

To set up a DB2 workload management configuration using monitoring data as the foundation:
1. Classify those applications that you know are important. You must isolate these applications and give them an appropriate portion of the system resources.
2. For the rest of the workload, collect statistics for the largest activities in the workload because these activities have the greatest impact on a per-activity basis on the system.
3. Analyze the activity information that you collected in step 2.
4. Repeat steps 1 through 3 on that portion of the workload that is still unclassified. Repeat this step until you know that the remaining unclassified work is not worth classification.

The sections that follow provide information about how to perform these steps.

Step 1. Isolate those applications that are known to be important and give them an appropriate portion of resources

Assume that you have two important business intelligence applications, BI1 and BI2 and that you need to minimize the response times for these applications. You can create workloads for these two applications and map them to a service class called MOSTIMPORTANT for which you can assign system resources.

On the AIX operating system, you use the AIX Workload Manager to create a service class called MOSTIMPORTANT, and give this service class a guaranteed set of resources.

On the DB2 data server, you create the required service classes and workloads:
CREATE SERVICE CLASS MOSTIMPORTANT OUTBOUND CORRELATOR 'MOSTIMPORTANT'
CREATE WORKLOAD BI1WORKLOAD APPLNAME ('BI1') SERVICE CLASS MOSTIMPORTANT
CREATE WORKLOAD BI2WORKLOAD APPLNAME ('BI2') SERVICE CLASS MOSTIMPORTANT

For the purposes of this example, assume that even after you account for the known applications, a significant portion of the system workload is unaccounted for. You therefore need to better understand and possibly control this workload.
Step 2. For the remaining unclassified workload, collect statistics for the largest activities in the workload

A long-running activity has a greater individual impact on the system than a short-running activity has because the long-running activity occupies system resources for a longer period of time. However, collecting information about a long-running activity imposes no greater overhead than would be imposed by collecting information on a short-running activity. As a result, the best way to collect information on the largest proportion of the workload is to collect information on the longest-running activities first.

Start collecting activity information by first deciding on an activity lifetime above which you collect activity information. You can simplify this task by choosing a portion of the unclassified activities to be collected, such as 30%, and then observing the activity lifetime histogram for these activities. Allow the system to run so that the statistics are updated, then run the WLM_COLLECT_STATS procedure to send the statistics to an active statistics event monitor.

Use the following query to obtain the activity lifetime histogram for the SYSDEFAULTUSERCLASS service class as a table that represents the proportion of the total activities that fell into each lifetime range. This query is written assuming that the database does not have multiple members.

```sql
WITH TOTAL AS (
    SELECT PARENTSERVICECLASSNAME, SERVICECLASSNAME, HIST.HISTOGRAM_TYPE, SUM(NUMBER_IN_BIN) AS NUMBER_IN_BIN
    FROM HISTOGRAMBIN_DB2STATISTICS AS HIST, SYSCAT.SERVICECLASSES SC
    WHERE HIST.SERVICE_CLASS_ID = SC.SERVICECLASSID
    AND HIST.TOP >= 0
    AND SC.PARENTSERVICECLASSNAME = 'SYSDEFAULTUSERCLASS'
    AND SC.SERVICECLASSNAME = 'SYSDEFAULTSUBCLASS'
    AND HIST.HISTOGRAM_TYPE = 'CoordActLifetime'
    GROUP BY PARENTSERVICECLASSNAME, SERVICECLASSNAME, HIST.HISTOGRAM_TYPE)
SELECT CAST(CAST(TOP AS DOUBLE) / 60000 AS DECIMAL(14,3)) AS TOP_IN_MINUTES,
       CAST(100 * CAST(SUM(HIST.NUMBER_IN_BIN) AS DOUBLE) / TOTAL.NUMBER_IN_BIN AS DECIMAL(4,2)) AS PERCENT_IN_BIN
FROM HISTOGRAMBIN_DB2STATISTICS AS HIST, SYSCAT.SERVICECLASSES SC, TOTAL
WHERE HIST.SERVICE_CLASS_ID = SC.SERVICECLASSID
    AND HIST.TOP >= 0
    AND TOTAL.NUMBER_IN_BIN > 0
    AND SC.PARENTSERVICECLASSNAME = 'SYSDEFAULTUSERCLASS'
    AND SC.SERVICECLASSNAME = 'SYSDEFAULTSUBCLASS'
    AND HIST.HISTOGRAM_TYPE = 'CoordActLifetime'
    AND TOTAL.PARENTSERVICECLASSNAME = SC.PARENTSERVICECLASSNAME
    AND TOTAL.SERVICECLASSNAME = SC.SERVICECLASSNAME
    AND TOTAL.HISTOGRAM_TYPE = HIST.HISTOGRAM_TYPE
    GROUP BY TOP, SC.PARENTSERVICECLASSNAME, SC.SERVICECLASSNAME, HIST.HISTOGRAM_TYPE, TOTAL.NUMBER_IN_BIN;
```

<table>
<thead>
<tr>
<th>TOP_IN_MINUTES</th>
<th>PERCENT_IN_BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.00</td>
</tr>
<tr>
<td>0.000</td>
<td>0.00</td>
</tr>
<tr>
<td>0.000</td>
<td>0.00</td>
</tr>
<tr>
<td>0.000</td>
<td>0.00</td>
</tr>
<tr>
<td>0.000</td>
<td>0.00</td>
</tr>
<tr>
<td>0.000</td>
<td>0.00</td>
</tr>
</tbody>
</table>
The following figure shows the results of the preceding query plotted as a graph:

![Activity lifetime histogram of unclassified activities](image)

*Figure 56. Activity lifetime histogram of unclassified activities*
In this example, 30% of the activities fall into the 101 minutes or greater lifetime range. To capture information about these activities, create an activity lifetime threshold of 100 minutes with the CONTINUE and COLLECT ACTIVITY DATA options as shown in the following example. If this threshold is violated, activity information is sent to an active activities event monitor.

```
CREATE THRESHOLD COLLECTLONGESTRUNNING30PERCENT
FOR SERVICE CLASS SYSDEFAULTSUBCLASS UNDER SYSDEFAULTUSERCLASS
ACTIVITIES ENFORCEMENT DATABASE ENABLE
WHEN ACTIVITYTOTALTIME > 100 MINUTES COLLECT ACTIVITY DATA CONTINUE
```

Allow the system to run so that data is collected.

Assuming that the overhead of collecting information on 30% of the longest-running activities is acceptable, you can let the data collection continue for a few hours or a few days. You can use the collected data to determine which users and applications produce the longest running of the 30% of the DML activities that are still unclassified. These activities might include some that are time critical. You might uncover some surprises, such as low-priority applications that run significant numbers of large activities. When you finish collecting and analyzing the data, you can drop the threshold.

**Step 3. Analyze the information about activities collected in the previous step**

You can analyze the information you collected about activities in the previous step according to the application that submitted them. You might specify the following query:

```
SELECT SUBSTR (APPL_NAME, 1,16) APPLICATION_NAME,
       AVG(TIMESTAMPDIFF(4, CHAR(TIME_COMPLETED - TIME_CREATED)))
       AS AVG_LIFETIME_MINUTES
       COUNT(*) AS ACTIVITY_COUNT
FROM ACTIVITY_DB2ACTIVITIES
GROUP BY APPL_NAME
ORDER BY APPL_NAME

APPLICATION_NAME AVG_LIFETIME_MINUTES ACTIVITY_COUNT
================ ============= ===============
MOSTLYSMALL1  120           21
MOSTLYSMALL2  110           15
UNIMPORTANTAPP 150        10213
```

An analysis of the activities according to the submitting application shows that a large number of the longest-running activities were submitted by the UNIMPORTANTAPP application, which is a relatively unimportant application. You can use a workload to isolate this application from the other unclassified applications and map it to a service class called BESTEFFORT, which receives resources only when all other activities have their resource needs met.

According to the preceding results, the remaining applications in the default service class appear to submit few large activities. You might find it worthwhile to repeat the process of collecting activities executing in the default service class without restricting the collection to long-running activities.
Step 4. Repeat steps 1 to 3 on that portion of the workload that is still unclassified until the remaining unclassified work is not worth classification

Now that you have the two important applications running in the MOSTIMPORTANT service class and the unimportant application running in the BESTEFFORT service class, much less work is running in the default user service class. In this situation, it might be inexpensive to collect information about every activity in this service class. Alternatively, you might not need to further subdivide the work and can stop here. Assume that you want to collect information about the remaining activities, in case the remaining workload contains surprises. You can accomplish this task by setting COLLECT ACTIVITY DATA for the default user service class and creating an activities event monitor:

```
ALTER SERVICE CLASS SYSDEFAULTSUBCLASS UNDER SYSDEFAULTUSERCLASS
COLLECT ACTIVITY DATA ON COORDINATOR WITHOUT DETAILS
```

Allow the system to run so that data is collected. You can analyze the results as in step 3.

```
SELECT SUBSTR (APPL_NAME,1,16) APPLICATION_NAME,
       AVG(TIMESTAMPDIFF(4, CHAR(TIME_COMPLETED - TIME_CREATED)))
       AS AVG_LIFETIME_MINUTES
       COUNT(*) AS ACTIVITY_COUNT
FROM ACTIVITY_DB2ACTIVITIES
GROUP BY APPL_NAME
ORDER BY APPL_NAME

APPLICATION_NAME AVG_LIFETIME_MINUTES ACTIVITY_COUNT
================ ============= ===============
MOSTLYSMALL1 5 1501
MOSTLYSMALL2 7 124
ONLYSMALL 2 10123
```

The results show that the ONLYSMALL application produces the majority of the unclassified activities. Because this application was not included in the results when you collected information about the largest activities, you can assume that ONLYSMALL did not produce any large queries during the period of data collection.
Chapter 8. Reference

Procedures and table functions

WLM_CANCEL_ACTIVITY - Cancel an activity

This procedure cancels a given activity. If the cancel takes place, an error message will be returned to the application that submitted the activity that was cancelled.

Syntax

```sql
WLM_CANCEL_ACTIVITY (application_handle, uow_id, activity_id)
```

The schema is SYSPROC.

Procedure parameters

**application_handle**

An input argument of type BIGINT that specifies the application handle whose activity is to be cancelled. If the argument is null, no activity will be found and an SQL4702N with SQLSTATE 5U035 is returned.

**uow_id**

An input argument of type INTEGER that specifies the unit of work ID of the activity that is to be cancelled. If the argument is null, no activity will be found and an SQL4702N with SQLSTATE 5U035 is returned.

**activity_id**

An input argument of type INTEGER that specifies the activity ID which uniquely identifies the activity within the unit of work that is to be cancelled. If the argument is null, no activity will be found and an SQL4702N with SQLSTATE 5U035 is returned.

Authorization

One of the following authorities is required to execute the routine:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

Default PUBLIC privilege

None

Example

An administrator can use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function to find the application handle, unit of work ID and activity ID of an activity. To cancel an activity with application handle 1, unit of work ID 2 and activity ID 3:
CALL WLM_CANCEL_ACTIVITY(1, 2, 3)

Usage notes
- If no activity can be found, an SQL4702N with SQLSTATE 5U035 is returned.
- If the activity cannot be cancelled because it not in the correct state (not initialized), an SQL4703N (reason code 1) with SQLSTATE 5U016 is returned.
- If the activity is successfully cancelled, an SQL4725N with SQLSTATE 57014 is returned to the cancelled application.
- If, at the time of the cancel, the coordinator is processing a request for a different activity or is idle, the activity is placed into CANCEL_PENDING state and will be cancelled when the coordinator processes the next request.

MON_CAPTURE_ACTIVITY_IN_PROGRESS procedure - Collect activity information for activities event monitor

The MON_CAPTURE_ACTIVITY_IN_PROGRESS procedure gathers information about a specified activity and writes the information to the active activities event monitor.

When you apply this procedure to an activity with child activities, the procedure recursively generates a record for each child activity. This information is collected and sent when you call the procedure; the procedure does not wait until the parent activity completes execution. The record of the activity in the event monitor is marked as a partial record.

Authorization

One of the following authorizations is required:
- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

Default PUBLIC privilege

None

Syntax

```
MON_CAPTURE_ACTIVITY_IN_PROGRESS(application_handle, uow_id, activity_id)
```

The schema is SYSPROC.

Procedure parameters

If you do not specify all of the following parameters, no activity is found, and an SQL0171N error with SQLSTATE 42815 is returned.

application_handle
- An input argument of type BIGINT that specifies the handle of the application whose activity information is to be captured.
An input argument of type INTEGER that specifies the unit of work ID of the activity whose information is to be captured.

**activity_id**
An input argument of type INTEGER that specifies the activity ID that uniquely identifies the activity within the unit of work whose information is to be captured.

**Usage notes**
- If there is no active activities event monitor, an SQL1633W error with SQLSTATE 01H53 is returned.
- Activity information is collected only on the coordinator member for the activity.

**Example**

Assume that a user complains that stored procedure MYSCHEMA.MYSLOWSTP seems to be running slower than usual. The administrator wants to investigate the cause of the slowdown. Investigating while the stored procedure is running is not practical, so the administrator decides to capture information about the stored procedure activity and all of the activities nested within it.

An event monitor for DB2 activities named DB2ACTIVITIES has been activated. The administrator uses the MON_GET_ACTIVITY function to obtain the application handle, unit of work ID, and activity ID for the call of this stored procedure. Assuming that the activity is identified by an application handle of 1, a unit of work ID of 2, and an activity ID of 3, the administrator can now issue the call to MON_CAPTURE_ACTIVITY_IN_PROGRESS as follows:

```
CALL MON_CAPTURE_ACTIVITY_IN_PROGRESS(1,2,3)
```

After the procedure is completed, the administrator can use the following table function to find out where the activity spent its time. The function retrieves the information from the DB2ACTIVITIES event monitor.

```sql
CREATE FUNCTION SHOWCAPTUREDACTIVITY(APPHNDL BIGINT, UOWID INTEGER, ACTIVITYID INTEGER) 
RETURNS TABLE (UOW_ID INTEGER, ACTIVITY_ID INTEGER, STMT_TEXT VARCHAR(40), LIFE_TIME DOUBLE) 
LANGUAGE SQL 
READS SQL DATA 
NO EXTERNAL ACTION 
DETERMINISTIC 
RETURN WITH RAH (LEVEL, APPL_ID, PARENT_UOW_ID, PARENT_ACTIVITY_ID, UOW_ID, ACTIVITY_ID, STMT_TEXT, ACT_EXEC_TIME) AS 
(SELECT 1, ROOT.APPL_ID, ROOT.PARENT_UOW_ID, 
ROOT.PARENT_ACTIVITY_ID, ROOT.UOW_ID, ROOT.ACTIVITY_ID, 
ROOTSTMT.STMT_TEXT, ACT_EXEC_TIME 
FROM ACTIVITY_DB2ACTIVITIES ROOT, ACTIVITYSTMT_DB2ACTIVITIES ROOTSTMT 
WHERE ROOT.APPL_ID = ROOTSTMT.APPL_ID AND ROOT.AGENT_ID = APPHNDL 
AND ROOT.UOW_ID = ROOTSTMT.UOW_ID AND ROOT.UOW_ID = UOWID 
AND ROOT.ACTIVITY_ID = ROOTSTMT.ACTIVITY_ID AND ROOT.ACTIVITY_ID = ACTIVITYID 
UNION ALL 
SELECT PARENT.LEVEL +1, CHILD.APPL_ID, CHILD.PARENT_UOW_ID, 
CHILD.PARENT_ACTIVITY_ID, CHILD.UOW_ID, 
CHILD.ACTIVITY_ID, CHILDSSTM.STMT_TEXT, CHILD.ACT_EXEC_TIME 
FROM RAH PARENT, ACTIVITY_DB2ACTIVITIES CHILD, ACTIVITYSTMT_DB2ACTIVITIES CHILDSSTM 
WHERE PARENT.APPL_ID = CHILD.APPL_ID AND 
CHILD.APPL_ID = CHILDSSTM.APPL_ID AND 
PARENT.UOW_ID = CHILD.PARENT_UOW_ID AND 
```

```sql```
CHILD.UOW_ID = CHILDSMT.UOW_ID AND 
PARENT.ACTIVITY_ID = CHILD.PARENT_ACTIVITY_ID AND 
CHILD.ACTIVITY_ID = CHILDSMT.ACTIVITY_ID AND 
PARENT.LEVEL < 64 
)
SELECT UOW_ID, ACTIVITY_ID, SUBSTR(STMT_TEXT,1,40), 
ACT_EXEC_TIME AS 
LIFE_TIME
FROM RAH

The following sample query uses the table function:
SELECT * FROM TABLE(SHOWCAPTUREDACTIVITY(1, 2, 3))
AS ACTS ORDER BY UOW_ID, ACTIVITY_ID

MON_COLLECT_STATS procedure - Collect and reset statistics for workload management objects and operating system resources

The MON_COLLECT_STATS procedure gathers or resets statistics for service classes, workloads, work classes, threshold queues, and operating system resources.

If there is an active statistics event monitor, the MON_COLLECT_STATS procedure writes the statistics it collects to the statistics event monitor.

**Authorization**

One of the following authorizations is required:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

**Default PUBLIC privilege**

None

**Syntax**

```sql
MON_COLLECT_STATS (wait, statistics_timestamp)
```

The schema is SYSPROC.

**Routine parameters**

**wait**

An optional input argument of type CHAR that specifies whether this procedure returns immediately after initiating a statistics collection and reset. If 'Y' is specified, then the procedure will not return until all statistics have been written and flushed to the statistics event monitor tables. Otherwise, the procedure will return immediately after initiating a statistics collection and reset.
An optional output argument of type TIMESTAMP that returns the timestamp value for the beginning of the statistics collection.

Usage notes

The MON_COLLECT_STATS procedure performs the same collection operation (send statistics to the active statistics event monitor) and reset operation that occur automatically on the interval defined by the wlm_collect_int database configuration parameter.

If you call the procedure while another collection and reset request is in progress (for example, while another invocation of the procedure is running or automatic collection is occurring), SQL1632W with SQLSTATE 01H53 is returned, and your new request is ignored.

In asynchronous mode, the MON_COLLECT_STATS procedure only starts the collection and reset process. The procedure might return to the caller before all statistics have been written to the active statistics event monitor. Depending on how quickly the statistics collection and reset occur, the call to the MON_COLLECT_STATS procedure (which is itself an activity) is counted in the statistics for either the prior collection interval or the new collection interval that has just started.

In synchronous mode, the MON_COLLECT_STATS procedure does not return until the statistics collection is complete and all statistics are written to the tables of any active statistics event monitors. The timestamp at which the statistics collection began is returned via the statistics_timestamp output parameter.

Examples

1. Call MON_COLLECT_STATS to initiate statistics collection and reset.
   CALL MON_COLLECT_STATS()
   The following is an example of output from this query:
   Return Status = 0
2. Call MON_COLLECT_STATS to collect and reset statistics, and return only when data has been written to statistics event monitor tables.
   CALL MON_COLLECT_STATS('Y', ::collect_timestamp)
   The following is an example of output from this query:
   Return Status = 0
3. Call MON_COLLECT_STATS to collect and reset statistics while another call is in progress.
   CALL MON_COLLECT_STATS()
   The following is an example of output from this query:
   SQL1632W The collect and reset statistics request was ignored because another collect and reset statistics request is already in progress.

WLM_GET_ACTIVITY_DETAILS - Return detailed information about a specific activity

This function returns detailed information about a specific activity identified by its application handle, unit of work ID, and activity ID. This information includes details about any thresholds that the activity has violated.
Note: This table function has been deprecated and replaced by “MON_GET_ACTIVITY_DETAILS table function - Return information about an activity as an XML document” on page 377.

This function returns basic statistics of one or more service subclasses.

Syntax

```sql
WLM_GET_ACTIVITY_DETAILS(
  application_handle,
  uow_id,
  activity_id,
  member)
```

The schema is SYSPROC.

Table function parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application_handle</td>
<td>An input argument of type BIGINT that specifies a valid application handle. If the argument is null, no rows are returned from this function. If the argument is null, an SQL171N error is returned.</td>
</tr>
<tr>
<td>uow_id</td>
<td>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. If the argument is null, an SQL171N error is returned.</td>
</tr>
<tr>
<td>activity_id</td>
<td>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. If the argument is null, an SQL171N error is returned.</td>
</tr>
<tr>
<td>member</td>
<td>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 a null value is specified, -1 is set implicitly.</td>
</tr>
</tbody>
</table>

Authorization

One of the following authorities is required to execute the routine:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

Default PUBLIC privilege

None

Example

Detailed information about an individual activity can be obtained by using the WLM_GET_ACTIVITY_DETAILS table function. This table function returns activity information as name-value pairs for each member. This example is restricted to
showing only an eleven member subset of the name-value pairs for each member for an activity identified by an application handle of 1, a unit of work ID of 1 and an activity ID of 5. For a complete list of name-value pairs, see Table 74 on page 374 and Table 75 on page 375.

```sql
SELECT SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       SUBSTR(NAME, 1, 20) AS NAME,
       SUBSTR(VALUE, 1, 30) AS VALUE
FROM TABLE(WLM_GET_ACTIVITY_DETAILS(1, 1, 5, -2)) AS ACTDETAIL
WHERE NAME IN ('APPLICATION_HANDLE',
                'COORD_PARTITION_NUM',
                'LOCAL_START_TIME',
                'UOW_ID',
                'ACTIVITY_ID',
                'PARENT_UOW_ID',
                'PARENT_ACTIVITY_ID',
                'ACTIVITY_TYPE',
                'NESTING_LEVEL',
                'INVOCATION_ID',
                'ROUTINE_ID')
ORDER BY PART
```

The following is an example of output from this query.

<table>
<thead>
<tr>
<th>PART</th>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>APPLICATION_HANDLE</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>COORD_PARTITION_NUM</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>LOCAL_START_TIME</td>
<td>2005-11-25-18.52.49.343000</td>
</tr>
<tr>
<td>0</td>
<td>UOW_ID</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>ACTIVITY_ID</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>PARENT_UOW_ID</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>PARENT_ACTIVITY_ID</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>ACTIVITY_TYPE</td>
<td>READ_DML</td>
</tr>
<tr>
<td>0</td>
<td>NESTING_LEVEL</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>INVOCATION_ID</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>ROUTINE_ID</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>APPLICATION_HANDLE</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>COORD_PARTITION_NUM</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>LOCAL_START_TIME</td>
<td>2005-11-25-18.52.49.598000</td>
</tr>
<tr>
<td>1</td>
<td>UOW_ID</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>ACTIVITY_ID</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>PARENT_UOW_ID</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>PARENT_ACTIVITY_ID</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>ACTIVITY_TYPE</td>
<td>READ_DML</td>
</tr>
<tr>
<td>1</td>
<td>NESTING_LEVEL</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>INVOCATION_ID</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>ROUTINE_ID</td>
<td>0</td>
</tr>
</tbody>
</table>

**Usage note**

An ACTIVITY_STATE of QUEUED means that the coordinator activity has made a RPC to the catalog member to obtain threshold tickets and has not yet received a response. Seeing this state might indicate that the activity has been queued by WLM or, over short periods of time, might just indicate that the activity is in the process of obtaining its tickets. To obtain a more accurate picture of whether or not the activity is really being queued, one can determine which agent is working on the activity (using the WLM_GET_SERVICE_CLASS_AGENTS table function) and find out whether this agent's event_object at the catalog member has a value of WLM_QUEUE.
### Information returned

**Table 73. Information returned for WLM_GET_ACTIVITY_DETAILS**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR(256)</td>
<td>Element name. See Table 74 and Table 75 on page 375 for possible values.</td>
</tr>
<tr>
<td>VALUE</td>
<td>VARCHAR(1024)</td>
<td>Element values. See Table 74 and Table 75 on page 375 for possible values.</td>
</tr>
</tbody>
</table>

**Table 74. Elements returned**

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVITY_ID</td>
<td>activity_id - Activity ID monitor element</td>
</tr>
<tr>
<td>ACTIVITY_STATE</td>
<td>activity_state - Activity state monitor element</td>
</tr>
<tr>
<td>ACTIVITY_TYPE</td>
<td>activity_type - Activity type monitor element</td>
</tr>
<tr>
<td>APPLICATION_HANDLE</td>
<td>application_handle - Application handle monitor element</td>
</tr>
<tr>
<td>COORD_PARTITION_NUM</td>
<td>coord_partition_num - Coordinator partition number monitor element</td>
</tr>
<tr>
<td>DATABASE_WORK_ACTION_SET_ID</td>
<td>If this activity has been mapped to a work action set that has been applied to the database, this column contains the ID of the work action set. This column contains 0 if the activity has not been mapped to a work action set that has been applied to the database.</td>
</tr>
<tr>
<td>DATABASE_WORK_CLASS_ID</td>
<td>If this activity has been mapped to a work action set that has been applied to the database, this column contains the ID of the work class of this activity. This column contains 0 if the activity has not been mapped to a work action set that has been applied to the database.</td>
</tr>
<tr>
<td>EFFECTIVE_ISOLATION</td>
<td>effective_isolation - Effective isolation monitor element</td>
</tr>
<tr>
<td>EFFECTIVE_LOCK_TIMEOUT</td>
<td>effective_lock_timeout - Effective lock timeout monitor element</td>
</tr>
<tr>
<td>EFFECTIVE_QUERY_DEGREE</td>
<td>effective_query_degree - Effective query degree monitor element</td>
</tr>
<tr>
<td>ENTRY_TIME</td>
<td>entry_time - Entry time monitor element</td>
</tr>
<tr>
<td>INVOCATION_ID</td>
<td>invocation_id - Invocation ID monitor element</td>
</tr>
<tr>
<td>LAST_REFERENCE_TIME</td>
<td>last_reference_time - Last reference time monitor element</td>
</tr>
<tr>
<td>LOCAL_START_TIME</td>
<td>local_start_time - Local start time monitor element</td>
</tr>
<tr>
<td>NESTING_LEVEL</td>
<td>nesting_level - Nesting level monitor element</td>
</tr>
<tr>
<td>PACKAGE_NAME</td>
<td>package_name - Package name monitor element</td>
</tr>
<tr>
<td>PACKAGE_SCHEMA</td>
<td>package_schema - Package schema monitor element</td>
</tr>
<tr>
<td>PACKAGE_VERSION_ID</td>
<td>package_version_id - Package version monitor element</td>
</tr>
<tr>
<td>PARENT_ACTIVITY_ID</td>
<td>parent_activity_id - Parent activity ID monitor element</td>
</tr>
<tr>
<td>PARENT_UOW_ID</td>
<td>Unique unit of work identifier within an application. Refers to the original unit of work this activity's parent activity started in. Returns an empty string if the activity has no parent activity or when at a remote member.</td>
</tr>
</tbody>
</table>
Table 74. Elements returned (continued)

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_COST_ESTIMATE</td>
<td>query_cost_estimate - Query cost estimate monitor element</td>
</tr>
<tr>
<td>ROUTINE_ID</td>
<td>routine_id - Routine ID monitor element</td>
</tr>
<tr>
<td>ROWS_FETCHED</td>
<td>rows_fetched - Rows fetched monitor element</td>
</tr>
<tr>
<td>ROWS_MODIFIED</td>
<td>rows_modified - Rows modified monitor element</td>
</tr>
<tr>
<td>SECTION_NUMBER</td>
<td>section_number - Section number monitor element</td>
</tr>
<tr>
<td>SERVICE_CLASS_ID</td>
<td>service_class_id - Service class ID monitor element</td>
</tr>
<tr>
<td>SERVICE_CLASS_WORK_ACTION_SET_ID</td>
<td>If this activity has been mapped to a work action set that has been applied to a service class, this column contains the ID of the work action set. This column contains 0 if the activity has not been mapped to a work action set that has been applied to a service class.</td>
</tr>
<tr>
<td>SERVICE_CLASS_WORK_CLASS_ID</td>
<td>If this activity has been mapped to a work action set that has been applied to a service class, this column contains the ID of the work class of this activity. This column contains 0 if the activity has not been mapped to a work action set that has been applied to a service class.</td>
</tr>
<tr>
<td>STMT_PKG_CACHE_ID</td>
<td>stmt_pkgcache_id - Statement package cache identifier monitor element</td>
</tr>
<tr>
<td>STMT_TEXT</td>
<td>stmt_text - SQL statement text monitor element</td>
</tr>
<tr>
<td>SYSTEM_CPU_TIME</td>
<td>system_cpu_time - System CPU time monitor element</td>
</tr>
<tr>
<td>UOW_ID</td>
<td>uow_id - Unit of work ID monitor element</td>
</tr>
<tr>
<td>USER_CPU_TIME</td>
<td>user_cpu_time - User CPU time monitor element</td>
</tr>
<tr>
<td>UTILITY_ID</td>
<td>utility_id - Utility ID monitor element</td>
</tr>
</tbody>
</table>

**Important:** The WLM_GET_ACTIVITY_DETAILS table function shows only the thresholds that are currently being applied to an activity.

The following elements are returned only if the corresponding thresholds apply to the activity.

Table 75. Elements returned if applicable

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVITYTOTALTIME_THRESHOLD_ID</td>
<td>activitytotaltime_threshold_id - Activity total time threshold ID monitor element</td>
</tr>
<tr>
<td>ACTIVITYTOTALTIME_THRESHOLD_VALUE</td>
<td>activitytotaltime_threshold_value - Activity total time threshold value monitor element</td>
</tr>
<tr>
<td>ACTIVITYTOTALTIME_THRESHOLD_VIOLATED</td>
<td>activitytotaltime_threshold_violated - Activity total time threshold violated monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_ID</td>
<td>concurrentdbcoordactivities_db_threshold_id - Concurrent database coordinator activities database threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_QUEUE</td>
<td>concurrentdbcoordactivities_db_threshold_queue - Concurrent database coordinator activities database threshold queued monitor element</td>
</tr>
<tr>
<td>Element Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_VALUE</td>
<td><code>concurrentdbcoordactivities_db_threshold_value</code> - Concurrent database coordinator activities database threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_VIOLATED</td>
<td><code>concurrentdbcoordactivities_db_threshold_violated</code> - Concurrent database coordinator activities database threshold violated monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_ID</td>
<td><code>concurrentdbcoordactivities_subclass_threshold_id</code> - Concurrent database coordinator activities service subclass threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_QUEUE</td>
<td>'Yes' indicates that the activity was queued by the CONCURRENTDBCOORDACTIVITIES_SUBCLASS threshold. 'No' indicates that the activity was not queued.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_VALUE</td>
<td><code>concurrentdbcoordactivities_subclass_threshold_value</code> - Concurrent database coordinator activities service subclass threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_VIOLATED</td>
<td>'Yes' indicates that the activity violated the CONCURRENTDBCOORDACTIVITIES_SUBCLASS threshold. 'No' indicates that the activity has not yet violated the threshold.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_ID</td>
<td>The ID of the CONCURRENTDBCOORDACTIVITIES_SUPERCLASS threshold that was applied to the activity.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_QUEUE</td>
<td>'Yes' indicates that the activity was queued by the CONCURRENTDBCOORDACTIVITIES_SUPERCLASS threshold. 'No' indicates that the activity was not queued.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_VALUE</td>
<td>The upper bound of the CONCURRENTDBCOORDACTIVITIES_SUPERCLASS threshold that was applied to the activity.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_VIOLATED</td>
<td>'Yes' indicates that the activity violated the CONCURRENTDBCOORDACTIVITIES_SUPERCLASS threshold. 'No' indicates that the activity has not yet violated the threshold.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_ID</td>
<td>The ID of the CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET threshold that was applied to the activity.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_QUEUE</td>
<td>'Yes' indicates that the activity was queued by the CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET threshold. 'No' indicates that the activity was not queued.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_VALUE</td>
<td>The upper bound of the CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET threshold that was applied to the activity.</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_VIOLATED</td>
<td>'Yes' indicates that the activity violated the CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET threshold. 'No' indicates that the activity has not yet violated the threshold.</td>
</tr>
</tbody>
</table>
Table 75. Elements returned if applicable (continued)

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCURRENTWORKLOADACTIVITIES _THRESHOLD_ID</td>
<td>The ID of the CONCURRENTWORKLOADACTIVITIES threshold that was applied to the activity.</td>
</tr>
<tr>
<td>CONCURRENTWORKLOADACTIVITIES _THRESHOLD_VALUE</td>
<td>The upper bound of the CONCURRENTWORKLOADACTIVITIES threshold that was applied to the activity.</td>
</tr>
<tr>
<td>CONCURRENTWORKLOADACTIVITIES _THRESHOLD_VIOLATED</td>
<td>'Yes' indicates that the activity violated the CONCURRENTWORKLOADACTIVITIES threshold. 'No' indicates that the activity has not yet violated the threshold.</td>
</tr>
<tr>
<td>ESTIMATEDSQLCOST_THRESHOLD_ID</td>
<td>estimatedsqlcost_threshold_id - Estimated SQL cost threshold ID monitor element</td>
</tr>
<tr>
<td>ESTIMATEDSQLCOST_THRESHOLD_VALUE</td>
<td>estimatedsqlcost_threshold_value - Estimated SQL cost threshold value monitor element</td>
</tr>
<tr>
<td>ESTIMATEDSQLCOST_THRESHOLD_VIOLATED</td>
<td>estimatedsqlcost_threshold_violated - Estimated SQL cost threshold violated monitor element</td>
</tr>
<tr>
<td>SQLROWSRETURNED_THRESHOLD_ID</td>
<td>sqlrowsreturned_threshold_id - SQL rows read returned threshold ID monitor element</td>
</tr>
<tr>
<td>SQLROWSRETURNED_THRESHOLD_VALUE</td>
<td>sqlrowsreturned_threshold_value - SQL rows read returned threshold value monitor element</td>
</tr>
<tr>
<td>SQLROWSRETURNED_THRESHOLD_VIOLATED</td>
<td>sqlrowsreturned_threshold_violated - SQL rows read returned threshold violated monitor element</td>
</tr>
<tr>
<td>SQLTEMPSPACE_THRESHOLD_ID</td>
<td>sqltempspace_threshold_id - SQL temporary space threshold ID monitor element</td>
</tr>
<tr>
<td>SQLTEMPSPACE_THRESHOLD_VALUE</td>
<td>sqltempspace_threshold_value - SQL temporary space threshold value monitor element</td>
</tr>
<tr>
<td>SQLTEMPSPACE_THRESHOLD_VIOLATED</td>
<td>sqltempspace_threshold_violated - SQL temporary space threshold violated monitor element</td>
</tr>
</tbody>
</table>

**MON_GET_ACTIVITY_DETAILS table function - Return information about an activity as an XML document**

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

The metrics are returned in an XML document.

**Authorization**

One of the following authorities is required to execute the routine:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority

**Default PUBLIC privilege**

None
Syntax

```sql
MON_GET_ACTIVITY_DETAILS (application_handle, uow_id, activity_id, member)
```

The schema is SYSPROC.

**Routine 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.

**Information returned**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_HANDLE</td>
<td>BIGINT</td>
<td>application_handle - Application handle</td>
</tr>
<tr>
<td>UOW_ID</td>
<td>INTEGER</td>
<td>uow_id - Unit of work ID</td>
</tr>
<tr>
<td>ACTIVITY_ID</td>
<td>INTEGER</td>
<td>activity_id - Activity ID</td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member</td>
</tr>
<tr>
<td>DETAILS</td>
<td>BLOB(8M)</td>
<td>XML document that contains activity details. See Table 77 on page 379 for a description of the elements in this document.</td>
</tr>
</tbody>
</table>

The following example shows the structure of the XML document that is returned in the DETAILS column.

```xml
<db2_activity_details xmlns="http://www.ibm.com/xmlns/prod/db2/mon" release="90700000">
  <member>0</member>
  <application_handle>70</application_handle>
  <activity_id>1</activity_id>
  <activity_state>IDLE</activity_state>
  <activity_type>READ_DML</activity_type>
  <uow_id>1</uow_id>
  ...
  <activity_metrics release="90700000">
```
For the full schema, see sqlib/misc/DB2MonRoutines.xsd. This document uses the following XML non-primitive type definitions:

```xml
<xs:simpleType name = "executable_id_type" >
  <xs:annotation>
    <xs:documentation>
      The binary Executable ID
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base = "xs:hexBinary" >
    <xs:maxLength value = "32" />
  </xs:restriction>
</xs:simpleType>
```

### Detailed metrics returned

**Table 77. Detailed metrics returned for MON_GET_ACTIVITYDETAILS**

<table>
<thead>
<tr>
<th>Element name</th>
<th>Data type</th>
<th>Description or corresponding monitor element</th>
</tr>
</thead>
<tbody>
<tr>
<td>active_col_vector_consumers_top</td>
<td>xs:long</td>
<td>active_col_vector_consumers_top - Active columnar vector consumers high watermark</td>
</tr>
<tr>
<td>active_col_vector_consumers</td>
<td>xs:long</td>
<td>active_col_vector_consumers - Active columnar vector consumers</td>
</tr>
<tr>
<td>active_hash_grpbys_top</td>
<td>xs:long</td>
<td>active_hash_grpbys_top - Active hash Group By operations high watermark</td>
</tr>
<tr>
<td>active_hash_grpbys</td>
<td>xs:long</td>
<td>active_hash_grpbys - Active hash GROUP BY operations monitor element</td>
</tr>
<tr>
<td>active_hash_joins_top</td>
<td>xs:long</td>
<td>active_hash_joins_top - Active hash joins operations high watermark</td>
</tr>
<tr>
<td>active_hash_joins</td>
<td>xs:long</td>
<td>active_hash_joins - Active hash joins monitor element</td>
</tr>
<tr>
<td>active_olap_funcs_top</td>
<td>xs:long</td>
<td>active_olap_funcs_top - Active OLAP functions operations high watermark</td>
</tr>
<tr>
<td>active_olap_funcs</td>
<td>xs:long</td>
<td>active_olap_funcs - Active OLAP Functions monitor element</td>
</tr>
<tr>
<td>active_peas_top</td>
<td>xs:long</td>
<td>active_peas_top - Active partial early aggregations operations high watermark</td>
</tr>
<tr>
<td>active_peas</td>
<td>xs:long</td>
<td>active_peas - Active partial early aggregations operations</td>
</tr>
<tr>
<td>active_peds_top</td>
<td>xs:long</td>
<td>active_peds_top - Active partial early distinct operations high watermark</td>
</tr>
<tr>
<td>active_peds</td>
<td>xs:long</td>
<td>active_peds - Active partial early distinct operations</td>
</tr>
<tr>
<td>active_sort_consumers_top</td>
<td>xs:long</td>
<td>active_sort_consumers_top - Active sort consumers high watermark</td>
</tr>
<tr>
<td>active_sort_consumers</td>
<td>xs:long</td>
<td>active_sort_consumers - Active sort consumers</td>
</tr>
<tr>
<td>active_sorts_top</td>
<td>xs:long</td>
<td>active_sorts_top - Active Sorts high watermark</td>
</tr>
<tr>
<td>active_sorts</td>
<td>xs:long</td>
<td>active_sorts - Active Sorts monitor element</td>
</tr>
<tr>
<td>activity_id</td>
<td>xs:nonNegativeInteger</td>
<td>activity_id - Activity ID</td>
</tr>
<tr>
<td>activity_state</td>
<td>xs:string</td>
<td>activity_state - Activity state</td>
</tr>
<tr>
<td>activity_type</td>
<td>xs:string</td>
<td>activity_type - Activity type</td>
</tr>
</tbody>
</table>
Table 77. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

<table>
<thead>
<tr>
<th>Element name</th>
<th>Data type</th>
<th>Description or corresponding monitor element</th>
</tr>
</thead>
<tbody>
<tr>
<td>activitytotaltime_threshold_id</td>
<td>xs:int</td>
<td>activitytotaltime_threshold_id - Activity total time threshold ID</td>
</tr>
<tr>
<td>activitytotaltime_threshold_value</td>
<td>xs:dateTime</td>
<td>activitytotaltime_threshold_value - Activity total time threshold value</td>
</tr>
<tr>
<td>activitytotaltime_threshold_violated</td>
<td>xs:short</td>
<td>activitytotaltime_threshold_violated - Activity total time threshold violated</td>
</tr>
<tr>
<td>agents_top</td>
<td>xs:long</td>
<td>agents_top - Number of Agents Created monitor element</td>
</tr>
<tr>
<td>aggsqtempspace_threshold_id</td>
<td>xs:int</td>
<td>aggsqtempspace_threshold_id - AggSQL temporary space threshold ID.</td>
</tr>
<tr>
<td>aggsqtempspace_threshold_value</td>
<td>xs:long</td>
<td>aggsqtempspace_threshold_value - AggSQL temporary space threshold value</td>
</tr>
<tr>
<td>aggsqtempspace_threshold_violated</td>
<td>xs:short</td>
<td>aggsqtempspace_threshold_violated - AggSQL temporary space threshold violated</td>
</tr>
<tr>
<td>application_handle</td>
<td>xs:nonNegativeInteger</td>
<td>application_handle - Application handle</td>
</tr>
<tr>
<td>audit_events_total</td>
<td>xs:nonNegativeInteger</td>
<td>audit_events_total - Total audit events</td>
</tr>
<tr>
<td>audit_file_write_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>audit_file_write_wait_time - Audit file write wait time</td>
</tr>
<tr>
<td>audit_file_writes_total</td>
<td>xs:nonNegativeInteger</td>
<td>audit_file_writes_total - Total Audit files written</td>
</tr>
<tr>
<td>audit_subsystem_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>audit_subsystem_wait_time - Audit subsystem wait time</td>
</tr>
<tr>
<td>audit_subsystem_waits_total</td>
<td>xs:nonNegativeInteger</td>
<td>audit_subsystem_waits_total - Total audit subsystem waits</td>
</tr>
<tr>
<td>client_acctng</td>
<td>xs:string(255)</td>
<td>CURRENT_CLIENT_ACCTNG special register</td>
</tr>
<tr>
<td>client_applname</td>
<td>xs:string(255)</td>
<td>CURRENT_CLIENT_APPLNAME special register</td>
</tr>
<tr>
<td>client_userid</td>
<td>xs:string(255)</td>
<td>CURRENT_CLIENT_USERID special register</td>
</tr>
<tr>
<td>client_wrkstnname</td>
<td>xs:string(255)</td>
<td>CURRENT_CLIENT_WRKSTNNAME special register</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_db_threshold_id</td>
<td>xs:int</td>
<td>concurrentdbcoordactivities_db_threshold_id - Concurrent database coordinator activities threshold ID</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_db_threshold_queued</td>
<td>xs:short</td>
<td>concurrentdbcoordactivities_db_threshold_queued - Concurrent database coordinator activities</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_db_threshold_value</td>
<td>xs:long</td>
<td>concurrentdbcoordactivities_db_threshold_value - Concurrent database coordinator activities</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_db_threshold_violated</td>
<td>xs:short</td>
<td>concurrentdbcoordactivities_db_threshold_violated - Concurrent database coordinator activities threshold violated</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_subclass_threshold_id</td>
<td>xs:int</td>
<td>concurrentdbcoordactivities_subclass_threshold_id - Concurrent database coordinator activities subclass threshold ID</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_subclass_threshold_queued</td>
<td>xs:short</td>
<td>concurrentdbcoordactivities_subclass_threshold_queued - Concurrent database coordinator activities subclass threshold queued</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_subclass_threshold_value</td>
<td>xs:long</td>
<td>concurrentdbcoordactivities_subclass_threshold_value - Concurrent database coordinator activities subclass threshold value</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_subclass_threshold_violated</td>
<td>xs:short</td>
<td>concurrentdbcoordactivities_subclass_threshold_violated - Concurrent database coordinator activities subclass threshold violated</td>
</tr>
<tr>
<td>Element name</td>
<td>Data type</td>
<td>Description or corresponding monitor element</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_superclass_threshold_id</td>
<td>xs:int</td>
<td>concurrentdbcoordactivities_superclass_threshold_id - Concurrent database coordinator activities superclass</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_superclass_threshold_queued</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>concurrentdbcoordactivities_superclass_threshold_queued - Concurrent database coordinator activities superclass threshold queued</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_superclass_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>concurrentdbcoordactivities_superclass_threshold_violated - Concurrent database coordinator activities superclass threshold violated</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_superclass_threshold_value</td>
<td>xs:long</td>
<td>concurrentdbcoordactivities_superclass_threshold_value - Concurrent database coordinator activities superclass threshold value</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_wl_was_threshold_id</td>
<td>xs:int</td>
<td>concurrentdbcoordactivities_wl_was_threshold_id - Concurrent database coordinator activities workload work action set threshold ID</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_wl_was_threshold_queue</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>concurrentdbcoordactivities_wl_was_threshold_queue - Concurrent database coordinator activities workload work action set threshold queued</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_wl_was_threshold_value</td>
<td>xs:long</td>
<td>concurrentdbcoordactivities_wl_was_threshold_value - Concurrent database coordinator activities workload work action set threshold value</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_wl_was_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>concurrentdbcoordactivities_wl_was_threshold_violated - Concurrent database coordinator activities workload work action set threshold violated</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_work_action_set_threshold_queued</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_queued - Concurrent database coordinator activities work action set threshold queued</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_work_action_set_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_violated - Concurrent database coordinator activities work action set threshold violated</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_work_action_set_threshold_value</td>
<td>xs:long</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_value - Concurrent database coordinator activities work action set threshold value</td>
</tr>
<tr>
<td>concurrentdbcoordactivities_work_action_set_threshold_id</td>
<td>xs:int</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_id - Concurrent database coordinator activities work action set threshold ID</td>
</tr>
<tr>
<td>concurrentworkloadactivities_threshold_id</td>
<td>xs:int</td>
<td>concurrentworkloadactivities_threshold_id - Concurrent workload activities threshold ID monitor element</td>
</tr>
<tr>
<td>concurrentworkloadactivities_threshold_value</td>
<td>xs:long</td>
<td>concurrentworkloadactivities_threshold_value - Concurrent workload activities threshold value monitor element</td>
</tr>
<tr>
<td>concurrentworkloadactivities_threshold_violated</td>
<td>xs:short</td>
<td>concurrentworkloadactivities_threshold_violated - Concurrent workload activities threshold violated monitor element</td>
</tr>
<tr>
<td>coord_member</td>
<td>xs:nonNegativeInteger</td>
<td>coord_member - Coordinator member</td>
</tr>
<tr>
<td>coord_stmt_exec_time</td>
<td>xs:nonNegativeInteger</td>
<td>coord_stmt_exec_time - Execution time for statement by coordinator agent</td>
</tr>
<tr>
<td>cputime_threshold_id</td>
<td>xs:int</td>
<td>cputime_threshold_id - CPU time threshold ID</td>
</tr>
<tr>
<td>cputime_threshold_value</td>
<td>xs:long</td>
<td>cputime_threshold_value - CPU time threshold value</td>
</tr>
<tr>
<td>Element name</td>
<td>Data type</td>
<td>Description or corresponding monitor element</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cpu_time_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>cpu_time_threshold_violated - CPU time threshold violated</td>
</tr>
<tr>
<td>cpu_timeinsc_threshold_id</td>
<td>xs:int</td>
<td>cpu_timeinsc_threshold_id - CPU time in service threshold ID</td>
</tr>
<tr>
<td>cpu_timeinsc_threshold_value</td>
<td>xs:long</td>
<td>cpu_timeinsc_threshold_value - CPU time in service threshold value</td>
</tr>
<tr>
<td>cpu_timeinsc_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>cpu_timeinsc_threshold_violated - CPU time in service threshold violated</td>
</tr>
<tr>
<td>database_work_action_set_id</td>
<td>xs:nonNegativeInteger</td>
<td>db_work_action_set_id - Database work action set ID</td>
</tr>
<tr>
<td>database_work_class_id</td>
<td>xs:nonNegativeInteger</td>
<td>db_work_class_id - Database work class ID</td>
</tr>
<tr>
<td>datataginsc_threshold_id</td>
<td>xs:int</td>
<td>datataginsc_threshold_id - Datataginsc in threshold identifier</td>
</tr>
<tr>
<td>datataginsc_threshold_value</td>
<td>xs:string(32)</td>
<td>datataginsc_threshold_value - Datataginsc in threshold value</td>
</tr>
<tr>
<td>datataginsc_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>datataginsc_threshold_violated - Datataginsc in threshold violated</td>
</tr>
<tr>
<td>datatagnotinsc_threshold_id</td>
<td>xs:int</td>
<td>datatagnotinsc_threshold_id - Datatagnotinsc not in threshold identifier</td>
</tr>
<tr>
<td>datatagnotinsc_threshold_value</td>
<td>xs:string(32)</td>
<td>datatagnotinsc_threshold_value - Datatagnotinsc not in threshold value</td>
</tr>
<tr>
<td>datatagnotinsc_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>datatagnotinsc_threshold_violated - Datatagnotinsc not in threshold violated</td>
</tr>
<tr>
<td>deadlocks</td>
<td>xs:nonNegativeInteger</td>
<td>deadlocks - Deadlocks detected</td>
</tr>
<tr>
<td>diaglog_write_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>diaglog_write_wait_time - Diag log write time</td>
</tr>
<tr>
<td>diaglog_writes_total</td>
<td>xs:nonNegativeInteger</td>
<td>diaglog_writes_total - Diag log total writes</td>
</tr>
<tr>
<td>direct_read_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>direct_read_reqs - Direct read requests</td>
</tr>
<tr>
<td>direct_read_time</td>
<td>xs:nonNegativeInteger</td>
<td>direct_read_time - Direct read time</td>
</tr>
<tr>
<td>direct_reads</td>
<td>xs:nonNegativeInteger</td>
<td>direct_reads - Direct reads from database</td>
</tr>
<tr>
<td>direct_write_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>direct_write_reqs - Direct write requests</td>
</tr>
<tr>
<td>direct_write_time</td>
<td>xs:nonNegativeInteger</td>
<td>direct_write_time - Direct write time</td>
</tr>
<tr>
<td>direct_writes</td>
<td>xs:nonNegativeInteger</td>
<td>direct_writes - Direct writes to database</td>
</tr>
<tr>
<td>disabled_peds</td>
<td>xs:long</td>
<td>disabled_peds - Disabled partial early distincts monitor element</td>
</tr>
<tr>
<td>eff_stmt_text</td>
<td>xs:string</td>
<td>eff_stmt_text - Effective statement text. The first 1024 characters of the concentrated statement text following any literal replacement done by the statement concentrator. Only present if the statement concentrator is enabled and this statement was altered by the statement concentrator.</td>
</tr>
<tr>
<td>effective_isolation</td>
<td>xs:string</td>
<td>effective_isolation - Effective isolation</td>
</tr>
<tr>
<td>effective_lock_timeout</td>
<td>xs:integer</td>
<td>effective_lock_timeout - Effective lock time-out</td>
</tr>
<tr>
<td>effective_query_degree</td>
<td>xs:integer</td>
<td>effective_query_degree - Effective query degree</td>
</tr>
<tr>
<td>entry_time</td>
<td>xs:dateTime</td>
<td>entry_time - Entry timeThe time that this activity arrived into the system.</td>
</tr>
<tr>
<td>Element name</td>
<td>Data type</td>
<td>Description or corresponding monitor element</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>estimatedsqlcost_threshold_id</td>
<td>xs:int</td>
<td>estimatedsqlcost_threshold_id - Estimated SQL cost threshold ID</td>
</tr>
<tr>
<td>estimatedsqlcost_threshold_value</td>
<td>xs:long</td>
<td>estimatedsqlcost_threshold_value - Estimated SQL cost threshold value</td>
</tr>
<tr>
<td>estimatedsqlcost_threshold_violated</td>
<td>xs:short</td>
<td>estimatedsqlcost_threshold_violated - Estimated SQL cost threshold violated</td>
</tr>
<tr>
<td>evmon_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>evmon_wait_time - Event monitor wait time</td>
</tr>
<tr>
<td>evmon_waits_total</td>
<td>xs:nonNegativeInteger</td>
<td>evmon_waits_total - Event monitor total waits</td>
</tr>
<tr>
<td>executable_id</td>
<td>executable_id_type</td>
<td>executable_id - Executable ID</td>
</tr>
<tr>
<td>fcm_message_recv_volume</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_message_recv_volume - FCM message recv volume</td>
</tr>
<tr>
<td>fcm_message_recv_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_message_recv_wait_time - FCM message recv wait time</td>
</tr>
<tr>
<td>fcm_message_recvs_total</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_message_recvs_total - FCM message recvs total</td>
</tr>
<tr>
<td>fcm_message_send_volume</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_message_send_volume - FCM message send volume</td>
</tr>
<tr>
<td>fcm_message_send_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_message_send_wait_time - FCM message send wait time</td>
</tr>
<tr>
<td>fcm_message_sends_total</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_message_sends_total - FCM message sends total</td>
</tr>
<tr>
<td>fcm_recv_volume</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_recv_volume - FCM recv volume</td>
</tr>
<tr>
<td>fcm_recv_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_recv_wait_time - FCM recv wait time</td>
</tr>
<tr>
<td>fcm_recsvs_total</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_recsvs_total - FCM recvs total</td>
</tr>
<tr>
<td>fcm_send_volume</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_send_volume - FCM send volume</td>
</tr>
<tr>
<td>fcm_send_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_send_wait_time - FCM send wait time</td>
</tr>
<tr>
<td>fcm_sends_total</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_sends_total - FCM sends total</td>
</tr>
<tr>
<td>fcm_tq_recv_volume</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_tq_recv_volume - FCM tablequeue recv volume</td>
</tr>
<tr>
<td>fcm_tq_recv_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_tq_recv_wait_time - FCM tablequeue recv wait time</td>
</tr>
<tr>
<td>fcm_tq_recvs_total</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_tq_recvs_total - FCM tablequeue recvs total</td>
</tr>
<tr>
<td>fcm_tq_send_volume</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_tq_send_volume - FCM tablequeue send volume</td>
</tr>
<tr>
<td>fcm_tq_send_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_tq_send_wait_time - FCM tablequeue send wait time</td>
</tr>
<tr>
<td>fcm_tq_sends_total</td>
<td>xs:nonNegativeInteger</td>
<td>fcm_tq_sends_total - FCM tablequeue sends total</td>
</tr>
<tr>
<td>hash_grpby_overflows</td>
<td>xs:long</td>
<td>hash_grpby_overflows - Hash group by overflows</td>
</tr>
<tr>
<td>hash_join_overflows</td>
<td>xs:long</td>
<td>hash_join_overflows - Hash Join Overflows monitor element</td>
</tr>
<tr>
<td>hash_join_small_overflows</td>
<td>xs:long</td>
<td>hash_join_small_overflows - Hash Join Small Overflows monitor element</td>
</tr>
<tr>
<td>idaRecv_volume</td>
<td>xs:nonNegativeInteger</td>
<td>The total volume of data the database server received from an in-database analytics process. The value is reported in bytes.</td>
</tr>
<tr>
<td>ida_recv_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>The total amount of time spent waiting to receive data from an in-database analytics process.</td>
</tr>
<tr>
<td>ida_recsvs_total</td>
<td>xs:nonNegativeInteger</td>
<td>The total number of times data was received from an in-database analytics process.</td>
</tr>
<tr>
<td>ida_send_volume</td>
<td>xs:nonNegativeInteger</td>
<td>The total volume of data sent from the database server to an in-database analytics process. The value is reported in bytes.</td>
</tr>
<tr>
<td>Element name</td>
<td>Data type</td>
<td>Description or corresponding monitor element</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ida_send_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>The total amount of time spent waiting to send data to an in-database analytics process.</td>
</tr>
<tr>
<td>ida_sends_total</td>
<td>xs:nonNegativeInteger</td>
<td>The total number of times data was sent to an in-database analytics process.</td>
</tr>
<tr>
<td>int_rows_deleted</td>
<td>xs:long</td>
<td>int_rows_deleted - Internal Rows Deleted monitor element</td>
</tr>
<tr>
<td>int_rows_inserted</td>
<td>xs:long</td>
<td>int_rows_inserted - Internal Rows Inserted monitor element</td>
</tr>
<tr>
<td>int_rows_updated</td>
<td>xs:long</td>
<td>int_rows_updated - Internal Rows Updated monitor element</td>
</tr>
<tr>
<td>intra_parallel_state</td>
<td>xs:string</td>
<td>intra_parallel_state - Current state of intrapartition parallelism monitor element</td>
</tr>
<tr>
<td>invocation_id</td>
<td>xs:nonNegativeInteger</td>
<td>stmt_invocation_id - Statement invocation identifier</td>
</tr>
<tr>
<td>is_system_appl</td>
<td>xs:short</td>
<td>is_system_appl - Is System Application monitor element</td>
</tr>
<tr>
<td>last_reference_time</td>
<td>xs:dateTime</td>
<td>last_reference_time - Last reference time. Every time a request occurs in this activity, this field is updated.</td>
</tr>
<tr>
<td>local_start_time</td>
<td>xs:dateTime</td>
<td>local_start_time - Local start time.</td>
</tr>
<tr>
<td>lock_escals</td>
<td>xs:nonNegativeInteger</td>
<td>lock_escals - Number of lock escalations</td>
</tr>
<tr>
<td>lock_timeouts</td>
<td>xs:nonNegativeInteger</td>
<td>lock_timeouts - Number of lock timeouts</td>
</tr>
<tr>
<td>lock_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>lock_wait_time - Time waited on locks</td>
</tr>
<tr>
<td>lock_waits</td>
<td>xs:nonNegativeInteger</td>
<td>lock_waits - Lock waits</td>
</tr>
<tr>
<td>log_buffer_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>log_buffer_wait_time - Log buffer wait time</td>
</tr>
<tr>
<td>log_disk_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>log_disk_wait_time - Log disk wait time</td>
</tr>
<tr>
<td>log_disk_waits_total</td>
<td>xs:nonNegativeInteger</td>
<td>log_disk_waits_total - Log disk waits total</td>
</tr>
<tr>
<td>member</td>
<td>xs:nonNegativeInteger</td>
<td>member - Database member</td>
</tr>
<tr>
<td>nesting_level</td>
<td>xs:nonNegativeInteger</td>
<td>stmt_nest_level - Statement nesting level</td>
</tr>
<tr>
<td>num_agents</td>
<td>xs:long</td>
<td>num_agents - Number of Agents Working on a Statement monitor element</td>
</tr>
<tr>
<td>num_log_buffer_full</td>
<td>xs:nonNegativeInteger</td>
<td>num_log_buffer_full - Number of full log buffers</td>
</tr>
<tr>
<td>num_lw_thresh_exceeded</td>
<td>xs:nonNegativeInteger</td>
<td>num_lw_thresh_exceeded - Number of thresholds exceeded</td>
</tr>
<tr>
<td>num_routines</td>
<td>xs:int</td>
<td>num_routines - Number of routines</td>
</tr>
<tr>
<td>olap_func_overflows</td>
<td>xs:long</td>
<td>olap_func_overflows - OLAP Function Overflows monitor element</td>
</tr>
<tr>
<td>package_name</td>
<td>xs:string (128)</td>
<td>package_name - Package name</td>
</tr>
<tr>
<td>package_schema</td>
<td>xs:string (128)</td>
<td>package_schema - Package schema</td>
</tr>
<tr>
<td>package_version_id</td>
<td>xs:string (128)</td>
<td>package_version_id - Package version</td>
</tr>
<tr>
<td>parent_activity_id</td>
<td>xs:nonNegativeInteger</td>
<td>parent_activity_id - Parent activity ID</td>
</tr>
<tr>
<td>parent_uow_id</td>
<td>xs:nonNegativeInteger</td>
<td>parent_uow_id - Parent unit of work ID</td>
</tr>
<tr>
<td>planid</td>
<td>xs:long</td>
<td>planid - Query plan ID monitor element</td>
</tr>
<tr>
<td>pool_col_gbp_indep_pages_found_in_lbp</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_gbp_indep_pages_found_in_lbp - Buffer pool column-organized GBP independent pages found in local buffer pool</td>
</tr>
<tr>
<td>Element name</td>
<td>Data type</td>
<td>Description or corresponding monitor element</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pool_col_gbp_invalid_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_gbp_invalid_pages - Buffer pool column-organized GBP invalid data pages</td>
</tr>
<tr>
<td>pool_col_gbp_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_gbp_l_reads - Buffer pool column-organized GBP logical reads</td>
</tr>
<tr>
<td>pool_col_gbp_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_gbp_p_reads - Buffer pool column-organized GBP physical reads</td>
</tr>
<tr>
<td>pool_col_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_l_reads - Buffer pool column-organized logical reads</td>
</tr>
<tr>
<td>pool_col_lbp_pages_found</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_lbp_pages_found - Buffer pool column-organized LBP pages found</td>
</tr>
<tr>
<td>pool_col_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_p_reads - Buffer pool column-organized physical reads</td>
</tr>
<tr>
<td>pool_col_writes</td>
<td>xs:nonNegativeInteger</td>
<td>pool_col_writes - Buffer pool column-organized writes</td>
</tr>
<tr>
<td>pool_data_gbp_indep_pages_found_in_lbp</td>
<td>xs:nonNegativeInteger</td>
<td>pool_data_gbp_indep_pages_found_in_lbp - Group buffer pool independent data pages found in local buffer pool monitor element</td>
</tr>
<tr>
<td>pool_data_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_data_l_reads - Buffer pool data logical reads</td>
</tr>
<tr>
<td>pool_data_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_data_p_reads - Buffer pool data physical reads</td>
</tr>
<tr>
<td>pool_data_writes</td>
<td>xs:nonNegativeInteger</td>
<td>pool_data_writes - Buffer pool data writes</td>
</tr>
<tr>
<td>pool_failed_async_col_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_failed_async_col_reqs - Failed column-organized async col reqs</td>
</tr>
<tr>
<td>pool_failed_async_data_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_failed_async_data_reqs - Failed data prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_failed_async_index_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_failed_async_index_reqs - Failed index prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_failed_async_other_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_failed_async_other_reqs - Failed non-prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_failed_async_temp_col_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_failed_async_temp_col_reqs - Failed column-organized temporary prefetch requests</td>
</tr>
<tr>
<td>pool_failed_async_temp_data_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_failed_async_temp_data_reqs - Failed data prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_failed_async_temp_index_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool.failed_async_temp_index_reqs - Failed index prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_failed_async_temp_xda_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool.failed_async_temp_xda_reqs - Failed XDA prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_index_gbp_indep_pages_found_in_lbp</td>
<td>xs:nonNegativeInteger</td>
<td>pool_index_gbp_indep_pages_found_in_lbp - Group buffer pool independent index pages found in local buffer pool monitor element</td>
</tr>
<tr>
<td>pool_index_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_index_l_reads - Buffer pool index logical reads</td>
</tr>
<tr>
<td>pool_index_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_index_p_reads - Buffer pool index physical reads</td>
</tr>
<tr>
<td>pool_index_writes</td>
<td>xs:nonNegativeInteger</td>
<td>pool_index_writes - Buffer pool index writes</td>
</tr>
<tr>
<td>pool_queued_async_col_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_col_pages - Column-organized page prefetch requests</td>
</tr>
<tr>
<td>Element name</td>
<td>Data type</td>
<td>Description or corresponding monitor element</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pool_queued_async_col_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_col_reqs - Column-organized prefetch requests</td>
</tr>
<tr>
<td>pool_queued_async_data_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_data_pages - Data pages prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_queued_async_data_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_data_reqs - Data prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_queued_async_index_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_index_pages - Index pages prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_queued_async_index_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_index_reqs - Index prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_queued_async_other_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_other_reqs - Non-prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_queued_async_temp_col_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_col_pages - Column-organized page temporary prefetch requests</td>
</tr>
<tr>
<td>pool_queued_async_temp_col_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_col_reqs - Column-organized temporary prefetch requests</td>
</tr>
<tr>
<td>pool_queued_async_temp_data_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_data_pages - Data pages prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_queued_async_temp_data_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_data_reqs - Data prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_queued_async_temp_index_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_index_pages - Index pages prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_queued_async_temp_index_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_index_reqs - Index prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_queued_async_temp_xda_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_xda_pages - XDA data pages prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_queued_async_temp_xda_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_temp_xda_reqs - XDA data prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>pool_queued_async_xda_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_xda_pages - XDA pages prefetch requests</td>
</tr>
<tr>
<td>pool_queued_async_xda_reqs</td>
<td>xs:nonNegativeInteger</td>
<td>pool_queued_async_xda_reqs - XDA prefetch requests monitor element</td>
</tr>
<tr>
<td>pool_read_time</td>
<td>xs:nonNegativeInteger</td>
<td>pool_read_time - Total buffer pool physical read time</td>
</tr>
<tr>
<td>pool_temp_col_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_col_l_reads - Buffer pool column-organized temporary logical reads</td>
</tr>
<tr>
<td>pool_temp_col_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_col_p_reads - Buffer pool column-organized temporary physical reads</td>
</tr>
<tr>
<td>pool_temp_data_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_data_l_reads - Buffer pool temporary data logical reads</td>
</tr>
<tr>
<td>pool_temp_data_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_data_p_reads - Buffer pool temporary data physical reads</td>
</tr>
<tr>
<td>pool_temp_index_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_index_l_reads - Buffer pool temporary index logical reads</td>
</tr>
<tr>
<td>pool_temp_index_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_index_p_reads - Buffer pool temporary index physical reads</td>
</tr>
<tr>
<td>Element name</td>
<td>Data type</td>
<td>Description or corresponding monitor element</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pool_temp_xda_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads</td>
</tr>
<tr>
<td>pool_temp_xda_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads</td>
</tr>
<tr>
<td>pool_write_time</td>
<td>xs:nonNegativeInteger</td>
<td>pool_write_time - Total buffer pool physical write time</td>
</tr>
<tr>
<td>pool_xda_gbp_indep_pages_found_in_lbp</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_gbp_indep_pages_found_in_lbp - Group buffer pool XDA independent pages found in local buffer pool</td>
</tr>
<tr>
<td>pool_xda_gbp_invalid_pages</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_gbp_invalid_pages - Group buffer pool invalid XDA data pages</td>
</tr>
<tr>
<td>pool_xda_gbp_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_gbp_l_reads - Group buffer pool XDA data logical read requests</td>
</tr>
<tr>
<td>pool_xda_gbp_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_gbp_p_reads - Group buffer pool XDA data physical read requests</td>
</tr>
<tr>
<td>pool_xda_l_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_l_reads - Buffer pool XDA data logical reads</td>
</tr>
<tr>
<td>pool_xda_lbp_pages_found</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_lbp_pages_found - Local buffer pool XDA data pages found</td>
</tr>
<tr>
<td>pool_xda_p_reads</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_p_reads - Buffer pool XDA data physical reads</td>
</tr>
<tr>
<td>pool_xda_writes</td>
<td>xs:nonNegativeInteger</td>
<td>pool_xda_writes - Buffer pool XDA data writes</td>
</tr>
<tr>
<td>post_shrthreshold_hash_joins</td>
<td>xs:long</td>
<td>post_shrthreshold_hash_joins - Post threshold hash joins monitor element</td>
</tr>
<tr>
<td>post_shrthreshold_sorts</td>
<td>xs:nonNegativeInteger</td>
<td>post_shrthreshold_sorts - Post shared threshold sorts</td>
</tr>
<tr>
<td>post_threshold_col_vector_consumers</td>
<td>xs:long</td>
<td>post_threshold_col_vector_consumers - Post threshold columnar vector consumers</td>
</tr>
<tr>
<td>post_threshold_hash_grpbys</td>
<td>xs:long</td>
<td>post_threshold_hash_grpbys - Hash group by threshold</td>
</tr>
<tr>
<td>post_threshold_hash_joins</td>
<td>xs:long</td>
<td>post_threshold_hash_joins - Hash Join Threshold monitor element</td>
</tr>
<tr>
<td>post_threshold_olap_funcs</td>
<td>xs:long</td>
<td>post_threshold_olap_funcs - OLAP function threshold monitor element</td>
</tr>
<tr>
<td>post_threshold_peas</td>
<td>xs:long</td>
<td>post_threshold_peas - Partial early aggregation threshold monitor element</td>
</tr>
<tr>
<td>post_threshold_peds</td>
<td>xs:long</td>
<td>post_threshold_peds - Partial early distincts threshold monitor element</td>
</tr>
<tr>
<td>post_threshold_sorts</td>
<td>xs:nonNegativeInteger</td>
<td>post_threshold_sorts - Post threshold sorts</td>
</tr>
<tr>
<td>prefetch_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>prefetch_wait_time - Time waited for prefetch</td>
</tr>
<tr>
<td>prefetch_waits</td>
<td>xs:nonNegativeInteger</td>
<td>prefetch_waits - Prefetcher wait count monitor element</td>
</tr>
<tr>
<td>query_actual_degree</td>
<td>xs:int</td>
<td>query_actual_degree - Actual runtime degree of intrapartition parallelism monitor element</td>
</tr>
<tr>
<td>query_card_estimate</td>
<td>xs:long</td>
<td>query_card_estimate - Query Number of Rows Estimate monitor element</td>
</tr>
<tr>
<td>query_cost_estimate</td>
<td>xs:integer</td>
<td>query_cost_estimate - Query cost estimate</td>
</tr>
<tr>
<td>query_data_tag_list</td>
<td>xs:string(32)</td>
<td>query_data_tag_list - Query data tag list</td>
</tr>
<tr>
<td>routine_id</td>
<td>xs:nonNegativeInteger</td>
<td>routine_id - Routine ID</td>
</tr>
</tbody>
</table>
Table 77. Detailed metrics returned for MON_GET_ACTIVITYDETAILS (continued)

<table>
<thead>
<tr>
<th>Element name</th>
<th>Data type</th>
<th>Description or corresponding monitor element</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows_deleted</td>
<td>xs:long</td>
<td>rows_deleted - Rows deleted monitor element</td>
</tr>
<tr>
<td>rows_inserted</td>
<td>xs:long</td>
<td>rows_inserted - Rows inserted monitor element</td>
</tr>
<tr>
<td>rows_modified</td>
<td>xs:nonNegativeInteger</td>
<td>rows_modified - Rows modified</td>
</tr>
<tr>
<td>rows_read</td>
<td>xs:nonNegativeInteger</td>
<td>rows_read - Rows read</td>
</tr>
<tr>
<td>rows_returned</td>
<td>xs:nonNegativeInteger</td>
<td>rows_returned - Rows returned</td>
</tr>
<tr>
<td>rows_updated</td>
<td>xs:long</td>
<td>rows_updated - Rows updated monitor element</td>
</tr>
<tr>
<td>section_number</td>
<td>xs:integer</td>
<td>section_number - Section number</td>
</tr>
<tr>
<td>semantic_env_id</td>
<td>xs:long</td>
<td>semantic_env_id - Query semantic compilation environment ID monitor element</td>
</tr>
<tr>
<td>service_class_id</td>
<td>xs:integer</td>
<td>service_class_id - Service class</td>
</tr>
<tr>
<td>service_class_work_action_set_id</td>
<td>xs:nonNegativeInteger</td>
<td>sc_work_action_set_id - Service class work action set ID</td>
</tr>
<tr>
<td>service_class_work_class_id</td>
<td>xs:nonNegativeInteger</td>
<td>sc_work_class_id - Service class work class ID</td>
</tr>
<tr>
<td>sort_consumer_heap_top</td>
<td>xs:long</td>
<td>sort_consumer_heap_top - Individual private sort heap consumer high watermark</td>
</tr>
<tr>
<td>sort_consumer_shrheap_top</td>
<td>xs:long</td>
<td>sort_consumer_shrheap_top - Individual shared sort heap consumer high watermark</td>
</tr>
<tr>
<td>sort_heap_allocated</td>
<td>xs:long</td>
<td>sort_heap_allocated - Total Sort Heap Allocated monitor element</td>
</tr>
<tr>
<td>sort_heap_top</td>
<td>xs:long</td>
<td>sort_heap_top - Sort private heap high watermark monitor element</td>
</tr>
<tr>
<td>sort_overflows</td>
<td>xs:nonNegativeInteger</td>
<td>sort_overflows - Sort overflows</td>
</tr>
<tr>
<td>sort_shrheap_allocated</td>
<td>xs:long</td>
<td>sort_shrheap_allocated - Sort Share Heap Currently Allocated monitor element</td>
</tr>
<tr>
<td>sort_shrheap_top</td>
<td>xs:long</td>
<td>sort_shrheap_top - Sort share heap high watermark monitor element</td>
</tr>
<tr>
<td>sqlrowsread_threshold_id</td>
<td>xs:int</td>
<td>sqlrowsread_threshold_ID - SQL rows read threshold ID</td>
</tr>
<tr>
<td>sqlrowsread_threshold_value</td>
<td>xs:long</td>
<td>sqlrowsread_threshold_value - SQL rows read threshold value</td>
</tr>
<tr>
<td>sqlrowsread_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>sqlrowsread_threshold_violated - SQL rows read threshold violated</td>
</tr>
<tr>
<td>sqlrowsreadinsc_threshold_id</td>
<td>xs:int</td>
<td>sqlrowsreadinsc_threshold_id - SQL rows read in service threshold ID</td>
</tr>
<tr>
<td>sqlrowsreadinsc_threshold_value</td>
<td>xs:long</td>
<td>sqlrowsreadinsc_threshold_value - SQL rows read in service threshold value</td>
</tr>
<tr>
<td>sqlrowsreadinsc_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>sqlrowsreadinsc_threshold_violated - SQL rows read in service threshold violated</td>
</tr>
<tr>
<td>sqlrowsreturned_threshold_id</td>
<td>xs:int</td>
<td>sqlrowsreturned_threshold_id - SQL rows read returned threshold ID</td>
</tr>
<tr>
<td>sqlrowsreturned_threshold_value</td>
<td>xs:long</td>
<td>sqlrowsreturned_threshold_value - SQL rows read returned threshold value</td>
</tr>
<tr>
<td>sqlrowsreturned_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>sqlrowsreturned_threshold_violated - SQL rows read returned threshold violated</td>
</tr>
<tr>
<td>sqltempspace_threshold_id</td>
<td>xs:int</td>
<td>sqltempspace_threshold_id - SQL temporary space threshold ID</td>
</tr>
</tbody>
</table>
### Table 77. Detailed metrics returned for MON_GET_ACTIVITY_DETAILS (continued)

<table>
<thead>
<tr>
<th>Element name</th>
<th>Data type</th>
<th>Description or corresponding monitor element</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqltempspace_threshold_value</td>
<td>xs:long</td>
<td>sqltempspace_threshold_value - SQL temporary space threshold value</td>
</tr>
<tr>
<td>sqltempspace_threshold_violated</td>
<td>xs:short (1 = yes, 0 = no)</td>
<td>sqltempspace_threshold_violated - SQL temporary space threshold violated</td>
</tr>
<tr>
<td>stmt_exec_time</td>
<td>xs:nonNegativeInteger</td>
<td>stmt_exec_time - Statement execution time</td>
</tr>
<tr>
<td>stmt_pkg_cache_id</td>
<td>xs:nonNegativeInteger</td>
<td>stmt_pkgcache_id - Statement package cache identifier</td>
</tr>
<tr>
<td>stmt_text</td>
<td>xs:string</td>
<td>stmt_text - SQL statement text. If the activity is dynamic SQL or it is static SQL for which the statement text is available, this field contains the first 1024 characters of the statement text. Otherwise, it contains an empty string.</td>
</tr>
<tr>
<td>stmtid</td>
<td>xs:long</td>
<td>stmtid - Query statement ID monitor element</td>
</tr>
<tr>
<td>stmtno</td>
<td>xs:int</td>
<td>stmtno - Statement number monitor element</td>
</tr>
<tr>
<td>thresh_violations</td>
<td>xs:nonNegativeInteger</td>
<td>thresh_violations - Number of threshold violations</td>
</tr>
<tr>
<td>total_act_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_act_time - Total activity time</td>
</tr>
<tr>
<td>total_act_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_act_wait_time - Total activity wait time</td>
</tr>
<tr>
<td>total_app_section_executions</td>
<td>xs:nonNegativeInteger</td>
<td>total_app_section_executions - Total section executions</td>
</tr>
<tr>
<td>total_col_executions</td>
<td>xs:nonNegativeInteger</td>
<td>total_col_executions - Total column-organized executions</td>
</tr>
<tr>
<td>total_col_proc_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_col_proc_time - Total column-organized processing time</td>
</tr>
<tr>
<td>total_col_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_col_time - Total column-organized time</td>
</tr>
<tr>
<td>total_col_vector_consumers</td>
<td>xs:long</td>
<td>total_col_vector_consumers - Total columnar vector consumers</td>
</tr>
<tr>
<td>total_cpu_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_cpu_time - Total CPU time</td>
</tr>
<tr>
<td>total_disp_run_queue_time</td>
<td>xs:long</td>
<td>total_disp_run_queue_time - Total dispatcher run queue time monitor element in microseconds</td>
</tr>
<tr>
<td>total_extended_latch_wait_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_extended_latch_wait_time - Total extended latch wait time monitor element</td>
</tr>
<tr>
<td>total_extended_latch_waits</td>
<td>xs:nonNegativeInteger</td>
<td>total_extended_latch_waits - Total extended latch waits monitor element</td>
</tr>
<tr>
<td>total_hash_grpbys</td>
<td>xs:long</td>
<td>total_hash_grpbys - Total hash group by operations</td>
</tr>
<tr>
<td>total_hash_joins</td>
<td>xs:long</td>
<td>total_hash_joins - Total Hash Joins monitor element</td>
</tr>
<tr>
<td>total_hash_loops</td>
<td>xs:long</td>
<td>total_hash_loops - Total Hash Loops monitor element</td>
</tr>
<tr>
<td>total_index_build_proc_time</td>
<td>xs:long</td>
<td>total_index_build_proc_time - Total non-wait time spent building indexes due to index creation</td>
</tr>
<tr>
<td>total_index_build_time</td>
<td>xs:long</td>
<td>total_index_build_time - Total time spent building indexes due to index creation</td>
</tr>
<tr>
<td>total_indexes_built</td>
<td>xs:long</td>
<td>total_indexes_built - Total number of indexes built</td>
</tr>
<tr>
<td>total_olap_funcs</td>
<td>xs:long</td>
<td>total_olap_funcs - Total OLAP Functions monitor element</td>
</tr>
<tr>
<td>total_peas</td>
<td>xs:long</td>
<td>total_peas - Total partial early aggregations monitor element</td>
</tr>
<tr>
<td>total_peds</td>
<td>xs:long</td>
<td>total_peds - Total partial early distincts monitor element</td>
</tr>
<tr>
<td>total_routine_invocations</td>
<td>xs:nonNegativeInteger</td>
<td>total_routine_invocations - Total routine invocations</td>
</tr>
<tr>
<td>total_routine_non_sect_proc_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_routine_non_sect_proc_time - Non-section processing time</td>
</tr>
</tbody>
</table>
### Table 77. Detailed metrics returned for MON_GET_ACTIVITYDETAILS (continued)

<table>
<thead>
<tr>
<th>Element name</th>
<th>Data type</th>
<th>Description or corresponding monitor element</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_routine_non_sect_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_routine_non_sect_time - Non-section routine execution time</td>
</tr>
<tr>
<td>total_routine_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_routine_time - Total routine time</td>
</tr>
<tr>
<td>total_routine_user_code_proc_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_routine_user_code_proc_time - Total routine user code processing time</td>
</tr>
<tr>
<td>total_routine_user_code_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_routine_user_code_time - Total routine user code time</td>
</tr>
<tr>
<td>total_section_proc_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_section_proc_time - Total section processing time</td>
</tr>
<tr>
<td>total_section_sort_proc_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_section_sort_proc_time - Total section sort processing time</td>
</tr>
<tr>
<td>total_section_sort_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_section_sort_time - Total section sort time</td>
</tr>
<tr>
<td>total_section_sorts</td>
<td>xs:nonNegativeInteger</td>
<td>total_section_sorts - Total section sorts.</td>
</tr>
<tr>
<td>total_section_time</td>
<td>xs:nonNegativeInteger</td>
<td>total_section_time - Total section time</td>
</tr>
<tr>
<td>total_sorts</td>
<td>xs:nonNegativeInteger</td>
<td>total_sorts - Total Sorts</td>
</tr>
<tr>
<td>tq_sort_heap_rejections</td>
<td>xs:long</td>
<td>tq_sort_heap_rejections - Table queue sort heap rejections monitor element</td>
</tr>
<tr>
<td>tq_sort_heap_requests</td>
<td>xs:long</td>
<td>tq_sort_heap_requests - Table queue sort heap requests monitor element</td>
</tr>
<tr>
<td>tq_tot_send_spills</td>
<td>xs:nonNegativeInteger</td>
<td>tq_tot_send_spills - Total number of tablequeue buffers overflowed</td>
</tr>
<tr>
<td>uow_id</td>
<td>xs:nonNegativeInteger</td>
<td>uow_id - Unit of work ID</td>
</tr>
<tr>
<td>utility_id</td>
<td>xs:nonNegativeInteger</td>
<td>utility_id - Utility ID</td>
</tr>
<tr>
<td>utility_invocation_id</td>
<td>xs:hexBinary</td>
<td>utility_invocation_id - Utility invocation ID monitor element</td>
</tr>
<tr>
<td>wl_work_action_set_id</td>
<td>xs:nonNegativeInteger</td>
<td>wl_work_action_set_id - Workload work action set identifier monitor element</td>
</tr>
<tr>
<td>wl_work_class_id</td>
<td>xs:nonNegativeInteger</td>
<td>wl_work_class_id - Workload work class identifier</td>
</tr>
<tr>
<td>wlm_queue_assignments_total</td>
<td>xs:nonNegativeInteger</td>
<td>wlm_queue_assignments_total - Workload manager total queue assignments</td>
</tr>
<tr>
<td>wlm_queue_time_total</td>
<td>xs:nonNegativeInteger</td>
<td>wlm_queue_time_total - Workload manager total queue time</td>
</tr>
</tbody>
</table>

### Usage notes

The MON_GET_ACTIVITYDETAILS 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.

**Example**

Investigate a long running query to determine if it is spending its time executing or waiting (for example, blocked on locks or I/O).

Note: The following queries can be combined into one statement and are shown in 2 steps for reasons of clarity. Also, if you want to retrieve the complete text, you could use the executable ID to obtain the statement text from the MON_GET_PKG_CACHE_STMT table function.

1. First use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function to list activities and their start times.

   ```sql
   SELECT application_handle, 
   activity_id, 
   uow_id, 
   local_start_time 
   FROM TABLE(
   WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES( 
   cast(NULL as bigint), -1)
   ) AS T
   The following is an example of output from this query.
   APPLICATION_HANDLE ACTIVITY_ID UOW_ID LOCAL_START_TIME
   --------------------------- ----------- ----------- --------------------------
   7 1 2 2008-06-10-10.06.55.675668
   16 1 7 2008-06-10-10.08.38.613610
   2 record(s) selected.
   ```

2. Then use the MON_GET_ACTIVITY_DETAILS table function to view the percentage of time that the activity has spent waiting.

   ```sql
   SELECT actmetrics.application_handle, 
   actmetrics.activity_id, 
   actmetrics.uow_id, 
   VARCHAR(actmetrics.stmt_text, 50) as stmt_text, 
   actmetrics.total_act_time, 
   actmetrics.total_act_wait_time, 
   CASE WHEN actmetrics.total_act_time > 0 
     THEN DEC(
       FLOAT(actmetrics.total_act_wait_time) / 
       FLOAT(actmetrics.total_act_time)) * 100, 5, 2) 
   ELSE NULL 
   END AS PERCENTAGE_WAIT_TIME 
   FROM TABLE(MON_GET_ACTIVITY_DETAILS(7, 2, 1, -2)) AS ACTDETAILS, 
   XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'), 
   'actmetrics/db2_activity_details' 
   PASSING XMLPARSE(DOCUMENT ACTDETAILS.DETAILS) as "actmetrics"
   ```
Use the MON_GET_ACTIVITY_DETAILS table function to create a query that captures information about all the activities currently running on a system.

- Example 1: Run the following command using the DB2 command line processor (CLP)

```sql
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,
total_act_time,
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
The following is an example of output from this query:

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>ACTIVITY_ID</th>
<th>UOW_ID</th>
<th>TOTAL_ACT_TIME</th>
<th>TOTAL_ACT_WAIT_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>5</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>17</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>49</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

SQL0445W Value "with A1 as (select * from table(wlm_get_workload_occurrence_activities(...
3 record(s) selected with 1 warning messages printed.

The following sample continues output from this query:

```sql
... STMT_TEXT
... select name from sysibm.systables
... select * from sysibm.systables
... with A1 as (select * from table(wlm_get_workload_occurrence_activities(null, -1))
... has been truncated. SQLSTATE=01004
3 record(s) selected with 1 warning messages printed.
```
**MON_GET_QUEUE_STATS** table function - Return threshold queue statistics

The **MON_GET_QUEUE_STATS** function returns basic statistics for one or more threshold queues on all active members. This function returns one row of statistics for each threshold queue.

**Authorization**

One of the following authorizations is required:
- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

**Default PUBLIC privilege**

None

**Syntax**

```sql
MON_GET_QUEUE_STATS (threshold_predicate, threshold_domain, threshold_name, threshold_id)
```

The schema is SYSPROC.

**Routine parameters**

*threshold_predicate*

An input argument of type VARCHAR(27) that specifies a threshold predicate. The possible values are as follows:

- **CONCDBC**
  - Concurrent database coordinator activities threshold
- **DBCONN**
  - Total database member connections threshold
- **SCCONN**
  - Total service class member connections threshold

If the argument is null or an empty string, data is returned for all thresholds that meet the other criteria.

The **threshold_predicate** values match those of the THRESHOLDPREDICATE column in the SYSCAT.THRESHOLDS view.

*threshold_domain*

An input argument of type VARCHAR(18) that specifies a threshold domain. The possible values are as follows:

- **DB** Database
- **SB** Service subclass
- **SP** Service superclass
WA Work action set

If the argument is null or an empty string, data is returned for all thresholds that meet the other criteria.

The threshold_domain values match those of the DOMAIN column in the SYSCAT.THRESHOLDS view.

threshold_name
An input argument of type VARCHAR(128) that specifies a threshold name. If the argument is null or an empty string, data is returned for all thresholds that meet the other criteria. The threshold_name values match those of the THRESHOLDNAME column in the SYSCAT.THRESHOLDS view.

threshold_id
An input argument of type INTEGER that specifies a threshold ID. If the argument is null or -1, data is returned for all thresholds that meet the other criteria. The threshold_id values match those of the THRESHOLDID column in the SYSCAT.THRESHOLDS view.

Information returned

Table 78. Information returned by the MON_GET_QUEUE_STATS routine

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRESHOLD_PREDICATE</td>
<td>VARCHAR(27)</td>
<td>threshold_predicate - Threshold predicate monitor element</td>
</tr>
<tr>
<td>THRESHOLD_DOMAIN</td>
<td>VARCHAR(18)</td>
<td>threshold_domain - Threshold domain monitor element</td>
</tr>
<tr>
<td>THRESHOLD_NAME</td>
<td>VARCHAR(128)</td>
<td>threshold_name - Threshold name monitor element</td>
</tr>
<tr>
<td>THRESHOLD_ID</td>
<td>INTEGER</td>
<td>thresholdid - Threshold ID monitor element</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>SERVICE_SUPERCLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>service_superclass_name - Service superclass name monitor element</td>
</tr>
<tr>
<td>SERVICE_SUBCLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>servicesubclass_name - Service subclass name monitor element</td>
</tr>
<tr>
<td>WORK_ACTION_SET_NAME</td>
<td>VARCHAR(128)</td>
<td>work_action_set_name - Work action set name monitor element</td>
</tr>
<tr>
<td>WORK_CLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>work_class_name - Work class name monitor element</td>
</tr>
<tr>
<td>WORKLOAD_NAME</td>
<td>VARCHAR(128)</td>
<td>workload_name - Workload name monitor element</td>
</tr>
<tr>
<td>LAST_RESET</td>
<td>TIMESTAMP</td>
<td>last_reset - Last Reset Timestamp monitor element</td>
</tr>
<tr>
<td>QUEUE_SIZE_TOP</td>
<td>INTEGER</td>
<td>queue_size_top - Queue size top monitor element</td>
</tr>
<tr>
<td>QUEUE_TIME_TOTAL</td>
<td>BIGINT</td>
<td>queue_time_total - Queue time total monitor element</td>
</tr>
<tr>
<td>QUEUE.Assignments_TOTAL</td>
<td>BIGINT</td>
<td>queue.assignments_total - Queue assignments total monitor element</td>
</tr>
</tbody>
</table>
Table 78. Information returned by the MON_GET_QUEUE_STATS routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE_SIZE_CURRENT</td>
<td>INTEGER</td>
<td>Number of connections or activities in the queue.</td>
</tr>
<tr>
<td>QUEUE_TIME_LATEST</td>
<td>BIGINT</td>
<td>Time spent in the queue by the last connection or activity to leave the queue. Units are milliseconds.</td>
</tr>
<tr>
<td>QUEUE_EXIT_TIME_LATEST</td>
<td>TIMESTAMP</td>
<td>Time that the last connection or activity left the queue.</td>
</tr>
<tr>
<td>THRESHOLD_CURRENT_CONCURRENCY</td>
<td>INTEGER</td>
<td>Number of connections or activities that are currently running according to the threshold.</td>
</tr>
<tr>
<td>THRESHOLD_MAX_CONCURRENCY</td>
<td>INTEGER</td>
<td>Maximum number of connections or activities that the threshold allows to be concurrently running.</td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member monitor element</td>
</tr>
</tbody>
</table>

**Usage notes**

The function does not aggregate data across queues (on a member) or across members (for one or more queues). However, you can use SQL queries to aggregate data, as shown in the example.

**Example**

The following query displays the basic statistics for all the queues on a system, across all members:

```sql
SELECT substr(THRESHOLD_NAME, 1, 6) THRESHNAME, THRESHOLD_PREDICATE, THRESHOLD_DOMAIN, MEMBER MEMB, QUEUE_SIZE_TOP, QUEUE_TIME_TOTAL, QUEUE_ASSIGNMENTS_TOTAL QUEUE_ASSIGN FROM table(MON_GET_QUEUE_STATS('', '', '', -1)) as QSTATS
```

Sample output is as follows:

```
THRESHNAME   THRESHOLD_PREDICATE   THRESHOLD_DOMAIN ...
------------- --------------------------- ------------------ ...
LIMIT1 CONCDBC DB ...
LIMIT2 SCCONN SP ...
LIMIT3 DBCONN DB ...
... MEMB QUEUE_SIZE_TOP QUEUE_TIME_TOTAL QUEUE_ASSIGN ...
... 0 12 1238540 734 ...
... 0 4 741249 24 ...
... 0 7 412785 128 ...
```

**MON_GET_AGENT table function - List agents running in a service class**

The MON_GET_AGENT function returns the list of agents, fenced mode processes (db2fmp processes), and system entities on a specified member that are running in a specified service class or on behalf of a specified application.
The system entities are non-agent threads and processes, such as page cleaners and prefetchers.

Refer to Table 79 on page 397 for a complete list of information that can be returned.

**Authorization**

One of the following authorizations is required:
- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

**Default PUBLIC privilege**

None

**Syntax**

```sql
MON_GET_AGENT(
  service_superclass_name,
  service_subclass_name,
  application_handle,
  member
)
```

The schema is SYSPROC.

**Routine parameters**

- `service_superclass_name`
  An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, data is retrieved for all the superclasses in the database.

- `service_subclass_name`
  An input argument of type VARCHAR(128) that refers to a specific subclass within a superclass. If the argument is null or an empty string, data is retrieved for all the subclasses in the database.

- `application_handle`
  An input argument of type BIGINT that specifies the application handle for which agent information is to be returned. If the argument is null, data is retrieved for all applications in the database. An application handle of 0 returns the system entities only.

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

**Table 79. Information returned by the MON_GET_AGENT routine**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_SUPERCLASS_NAME</td>
<td>VARCHAR (128)</td>
<td>service_superclass_name - Service superclass name monitor element</td>
</tr>
<tr>
<td>SERVICE_SUBCLASS_NAME</td>
<td>VARCHAR (128)</td>
<td>service_subclass_name - Service subclass name monitor element</td>
</tr>
<tr>
<td>APPLICATION_HANDLE</td>
<td>BIGINT</td>
<td>application_handle - Application handle monitor element</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>ENTITY</td>
<td>VARCHAR (32)</td>
<td>One of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the type of entity is an agent, the value is db2agent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If the type of entity is a fenced mode process, the value is db2fmp (pid) where pid is the process ID of the fenced mode process.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Otherwise, the value is the name of the system entity.</td>
</tr>
<tr>
<td>WORKLOAD_NAME</td>
<td>VARCHAR (128)</td>
<td>workload_name - Workload name monitor element</td>
</tr>
<tr>
<td>WORKLOAD_OCCURRENCE_ID</td>
<td>INTEGER</td>
<td>workload_occurrence_id - Workload occurrence identifier monitor element</td>
</tr>
<tr>
<td>UOW_ID</td>
<td>INTEGER</td>
<td>uow_id - Unit of work ID monitor element</td>
</tr>
<tr>
<td>ACTIVITY_ID</td>
<td>INTEGER</td>
<td>activity_id - Activity ID monitor element</td>
</tr>
<tr>
<td>PARENT_UOW_ID</td>
<td>INTEGER</td>
<td>parent_uow_id - Parent unit of work ID monitor element</td>
</tr>
<tr>
<td>PARENT_ACTIVITY_ID</td>
<td>INTEGER</td>
<td>parent_activity_id - Parent activity ID monitor element</td>
</tr>
<tr>
<td>AGENT_TID</td>
<td>BIGINT</td>
<td>agent_tid - Agent thread ID monitor element</td>
</tr>
<tr>
<td>AGENT_TYPE</td>
<td>VARCHAR (32)</td>
<td>Agent type. The agent types are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COORDINATOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OTHER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PDBSUBAGENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SMPSUBAGENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the value is COORDINATOR, the agent ID might change in concentrator environments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The SMPSUBAGENT type is returned for agents that are processing column-organized tables.</td>
</tr>
<tr>
<td>SMP_COORDINATOR</td>
<td>INTEGER</td>
<td>Indication of whether the agent is an SMP coordinator: 1 for yes and 0 for no.</td>
</tr>
<tr>
<td>AGENT_SUBTYPE</td>
<td>VARCHAR (32)</td>
<td>Agent subtype. The possible subtypes are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- DSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OTHER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- RPC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COLUMNAR</td>
</tr>
<tr>
<td>AGENT_STATE</td>
<td>VARCHAR (32)</td>
<td>Indication of whether an agent is associated or active. The possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ASSOCIATED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ACTIVE</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| EVENT_TYPE       | VARCHAR (32)| Type of event last processed by this agent. The possible values are as follows:  
|                  |             | - ACQUIRE  
|                  |             | - PROCESS  
|                  |             | - WAIT     
|                  |             | See Table 80 on page 401 for more information about possible values for this column. |
| EVENT_OBJECT     | VARCHAR (32)| Object of the event last processed by this agent. The possible values are as follows:  
|                  |             | - COMPRESSION_DICTIONARY_BUILD  
|                  |             | - IMPLICIT_REBIND  
|                  |             | - INDEX_RECREATE  
|                  |             | - LOCK  
|                  |             | - LOCK_ESCALATION  
|                  |             | - QP_QUEUE  
|                  |             | - REMOTE_REQUEST  
|                  |             | - REQUEST  
|                  |             | - ROUTINE  
|                  |             | - WLM_QUEUE  
|                  |             | See Table 80 on page 401 for more information about possible values for this column. |
| EVENT_STATE      | VARCHAR (32)| State of the event last processed by this agent. The possible values are as follows:  
|                  |             | - EXECUTING  
|                  |             | - IDLE     
<p>|                  |             | See Table 80 on page 401 for more information about possible values for this column. |
| REQUEST_ID       | VARCHAR (64)| Request ID. This value is unique only in combination with the value of application_handle. You can use this combination to distinguish between one request that is taking a long time and multiple requests; for example, to distinguish between one long fetch and multiple fetches. |</p>
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_TYPE</td>
<td>VARCHAR (32)</td>
<td>Type of request. The possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For coordinator agents:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– CLOSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– COMMIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– COMPILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– DESCRIBE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– EXCSQLEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– EXECIMMD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– EXECUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– FETCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– INTERNAL number, where number is the value of the internal constant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– OPEN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– PREPARE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– REBIND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– REDISTRIBUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– REORG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– ROLLBACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– RUNSTATS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For subagents with an AGENT_SUBTYPE column value of DSS, SMP, or COLUMNAR:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– If the subsection number is nonzero, the subsection number in the form SUBSECTION:subsection_number; otherwise, NULL.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| REQUEST_TYPE (continued) | VARCHAR (32) | • For subagents with an AGENT_SUBTYPE of RPC:  
  – ABP  
  – CATALOG  
  – INTERNAL  
  – REORG  
  – RUNSTATS  
  – WLM  
• For subagents with a SUBTYPE of OTHER:  
  – ABP  
  – APP_RBSVPT  
  – APP_RELSQLPT  
  – BACKUP  
  – CLOSE  
  – EXTERNAL_RBSVPT  
  – EVMON  
  – FORCE  
  – FORCE_ALL  
  – INTERNAL number, where number is the value of the internal constant  
  – INTERRUPT  
  – NOOP (if there is no request)  
  – QP  
  – REDISTRIBUTE  
  – STMT_RBSVPT  
  – STOP_USING  
  – UPDATE_DBM_CFG  
  – WLM |
| NESTING_LEVEL       | INTEGER       | nesting_level - Nesting level monitor element                               |
| INVOCATION_ID       | INTEGER       | invocation_id - Invocation ID monitor element                               |
| ROUTINE_ID          | INTEGER       | routine_id - Routine ID monitor element                                      |
| EVENT_OBJECT_NAME   | VARCHAR (1024)| Event object name. If the value of EVENT_OBJECT is LOCK, the value of this column is the name of the lock that the agent is waiting on. If the value of EVENT_OBJECT is WLM_QUEUE, the value of the column is the name of the WLM threshold that the agent is queued on. Otherwise, the value is NULL. |
| APPLICATION_NAME    | VARCHAR (128) | appl_name - Application name                                                 |
| APPLICATION_ID      | VARCHAR (128) | appl_id - Application ID                                                     |
| CLIENT_PID          | BIGINT        | client_pid - Client process ID                                               |
| SESSION_AUTH_ID     | VARCHAR (128) | session_auth_id - Session authorization ID                                   |
| REQUEST_START_TIME  | TIMESTAMP     | Time that the agent started processing the request on which it is currently working |
Table 79. Information returned by the MON_GET_AGENT routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT_STATE_LAST_UPDATE_TIME</td>
<td>TIMESTAMP</td>
<td>The last time that the event, being processed by the agent, was changed. The event currently processed by the agent is identified by the EVENT_TYPE, EVENT_OBJECT, and EVENT_STATE columns.</td>
</tr>
<tr>
<td>EXECUTABLE_ID</td>
<td>VARCHAR (32)</td>
<td>executable_id - Executable ID monitor element</td>
</tr>
<tr>
<td>FOR BIT DATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member monitor element</td>
</tr>
<tr>
<td>UTILITY_INVOCATION_ID</td>
<td>VARCHAR (32)</td>
<td>utility_invocation_id - Utility invocation ID</td>
</tr>
<tr>
<td>FOR BIT DATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IS_SYSTEM_APPL</td>
<td>SMALLINT</td>
<td>is_system_appl - Is System Application monitor element</td>
</tr>
</tbody>
</table>

Note: The possible combinations of EVENT_STATE, EVENT_TYPE, EVENT_OBJECT, and EVENT_OBJECT_NAME column values are listed in the following table.

Table 80. Possible combinations for EVENT_STATE, EVENT_TYPE, EVENT_OBJECT, and EVENT_OBJECT_NAME column values

<table>
<thead>
<tr>
<th>Event description</th>
<th>EVENT_STATE value</th>
<th>EVENT_TYPE value</th>
<th>EVENT_OBJECT value</th>
<th>EVENT_OBJECT_NAME value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquire lock</td>
<td>IDLE</td>
<td>ACQUIRE</td>
<td>LOCK</td>
<td>Lock name</td>
</tr>
<tr>
<td>Escalate lock</td>
<td>EXECUTING</td>
<td>PROCESS</td>
<td>LOCK_ESCALATION</td>
<td>NULL</td>
</tr>
<tr>
<td>Process request</td>
<td>EXECUTING</td>
<td>PROCESS</td>
<td>REQUEST</td>
<td>NULL</td>
</tr>
<tr>
<td>Wait for a new request</td>
<td>IDLE</td>
<td>WAIT</td>
<td>REQUEST</td>
<td>NULL</td>
</tr>
<tr>
<td>Wait for a request to be processed at a remote member</td>
<td>IDLE</td>
<td>WAIT</td>
<td>REMOTE_REQUEST</td>
<td>NULL</td>
</tr>
<tr>
<td>Wait on a WLM threshold queue</td>
<td>IDLE</td>
<td>WAIT</td>
<td>WLM_QUEUE</td>
<td>Threshold name</td>
</tr>
<tr>
<td>Process a routine</td>
<td>EXECUTING</td>
<td>PROCESS</td>
<td>ROUTINE</td>
<td>NULL</td>
</tr>
<tr>
<td>Re-create an index</td>
<td>EXECUTING</td>
<td>PROCESS</td>
<td>INDEX_RECREATE</td>
<td>NULL</td>
</tr>
<tr>
<td>Build compression dictionary</td>
<td>EXECUTING</td>
<td>PROCESS</td>
<td>COMP_DICT_BLD</td>
<td>NULL</td>
</tr>
<tr>
<td>Implicit rebind</td>
<td>EXECUTING</td>
<td>PROCESS</td>
<td>IMPLICIT_REBIND</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Usage notes

The parameters are, in effect, ANDed together. That is, if you specify conflicting input parameters, such as a service superclass SUP_A and a subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows are returned.

Examples

1. The following query returns a list of agents that are associated with application handle 1 for all database members. You can determine the application handle by using the LIST APPLICATIONS command or the MON_GET_CONNECTION table function.
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
    SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
    SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
    SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
    SUBSTR(AGENT_STATE,1,10) AS AGENTSTATE,
    SUBSTR(REQUEST_TYPE,1,12) AS REQTYPE,
    SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
    SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
FROM TABLE(MON_GET_AGENT(CAST(NULL AS VARCHAR(128)),
    CAST(NULL AS VARCHAR(128)), 1, -2)) AS SCDETAILS
ORDER BY APPHANDLE, MEMB, AGENT_TID

Sample output is as follows:

<table>
<thead>
<tr>
<th>APPHANDLE</th>
<th>MEMB</th>
<th>AGENT_TID</th>
<th>AGENTTYPE</th>
<th>AGENTSTATE</th>
<th>REQTYPE</th>
<th>UOW_ID</th>
<th>ACT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>COORDINATOR</td>
<td>ACTIVE</td>
<td>FETCH</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>SUBAGENT</td>
<td>ACTIVE</td>
<td>SUBSECTION:1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>SUBAGENT</td>
<td>ACTIVE</td>
<td>SUBSECTION:2</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

The output shows a coordinator agent and a subagent on member 0 and a subagent on member 1 operating on behalf of an activity with UOW ID 1 and activity ID 5. The AGENTTYPE column with a value of COORDINATOR has a value of FETCH for the REQTYPE column (which indicates the main or initial request type). This means that the type of request is a fetch request for the coordinator agent.

2. The following query determines which lock an agent is waiting on:

```sql
select event_object, event_type, event_state, varchar(event_object_name, 30) as event_object_name
from table(MON_GET_AGENT('', '', cast(NULL as bigint), -1)) as t
```

Sample output is as follows:

<table>
<thead>
<tr>
<th>EVENT_OBJECT</th>
<th>EVENT_TYPE</th>
<th>EVENT_STATE</th>
<th>EVENT_OBJECT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
</tbody>
</table>

21 record(s) selected.

Using the same query at a later time shows that the WLM threshold has queued an agent:

```sql
select event_object, event_type, event_state, varchar(event_object_name, 30) as event_object_name
from table(MON_GET_AGENT('', '', cast(NULL as bigint), -1)) as t
```

Sample output is as follows:

<table>
<thead>
<tr>
<th>EVENT_OBJECT</th>
<th>EVENT_TYPE</th>
<th>EVENT_STATE</th>
<th>EVENT_OBJECT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
<tr>
<td>REQUEST</td>
<td>PROCESS</td>
<td>EXECUTING</td>
<td>-</td>
</tr>
</tbody>
</table>

21 record(s) selected.
WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES - list workload occurrences

The WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES function returns the list of all workload occurrences running in a specified service class on a particular member. A workload occurrence is a specific database connection whose attributes match the definition of a workload and hence is associated with or assigned to the workload.

Refer to Table 81 on page 405 for a complete list of information that can be returned.

Syntax

```
WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES (service_superclass_name, service_subclass_name, member)
```

The schema is SYSPROC.

Table function parameters

`service_superclass_name`
An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, the data is retrieved for all the superclasses in the database that match the values of the other parameters.

`service_subclass_name`
Target service subclass for the workload occurrence. Any work submitted by this workload occurrence will run in this service subclass under the target service superclass with the exception of activities that are mapped, or remapped, to a different subclass.

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

One of the following authorities is required to execute the routine:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

Default PUBLIC privilege

None

Example

If an administrator wants to see what workload occurrences are running on the system as a whole, the administrator can call the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES function by specifying a null value or an empty string for service_superclass_name and service_subclass_name and -2 for member:

```sql
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(COORD_MEMBER),1,4) AS COORDMEMB,
       SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHNDL,
       SUBSTR(WORKLOAD_NAME,1,22) AS WORKLOAD_NAME,
       SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES
            (CAST(NULL AS VARCHAR(128)), CAST(NULL AS VARCHAR(128)), -2))
       AS SCINFO
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, MEMB, APPHNDL,
         WORKLOAD_NAME, WLO_ID
```

If the system has four database members and is currently running two workloads, the previous query produces results such as the following ones:

```
SUPERCLASS_NAME SUBCLASS_NAME MEMB COORDMEMB ...
------------------- ------------------ ---- --------- ...
SYSDEFAULTMAINTENAN SYSDEFAULTSUBCLASS 0 0 ...
SYSDEFAULTSYSTEMCLA SYSDEFAULTSUBCLASS 0 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3 0 ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3 0 ...
...
APPHNDL WORKLOAD_NAME WLO_ID ...
...
...
...
...
...
...
```
Usage note

The parameters are, in effect, ANDed together. That is, if you specify conflicting input parameters, such as a service superclass SUP_A and a subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows are returned.

Note: Statistics reported for the workload occurrence (for example, coord_act_completed_total) are reset at the beginning of each unit of work when they are combined with the corresponding workload statistics.

Information returned

Table 81. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_SUPERCLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>service_superclass_name - Service superclass name monitor element</td>
</tr>
<tr>
<td>SERVICE_SUBCLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>service_subclass_name - Service subclass name monitor element</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>COORD_PARTITION_NUM</td>
<td>SMALLINT</td>
<td>coord_partition_num - Coordinator partition number monitor element</td>
</tr>
<tr>
<td>APPLICATION_HANDLE</td>
<td>BIGINT</td>
<td>application_handle - Application handle monitor element</td>
</tr>
<tr>
<td>WORKLOAD_NAME</td>
<td>VARCHAR(128)</td>
<td>workload_name - Workload name monitor element</td>
</tr>
<tr>
<td>WORKLOAD_OCCURRENCE_ID</td>
<td>INTEGER</td>
<td>workload_occurrence_id - Workload occurrence identifier monitor element</td>
</tr>
<tr>
<td>UOW_ID</td>
<td>INTEGER</td>
<td>uow_id - Unit of work ID monitor element</td>
</tr>
<tr>
<td>WORKLOAD_OCCURRENCE_STATE</td>
<td>VARCHAR(32)</td>
<td>workload_occurrence_state - Workload occurrence state monitor element</td>
</tr>
<tr>
<td>SYSTEM_AUTH_ID</td>
<td>VARCHAR(128)</td>
<td>system_auth_id - System authorization identifier monitor element</td>
</tr>
<tr>
<td>SESSION_AUTH_ID</td>
<td>VARCHAR(128)</td>
<td>session_auth_id - Session authorization ID monitor element</td>
</tr>
<tr>
<td>APPLICATION_NAME</td>
<td>VARCHAR(128)</td>
<td>appl_name - Application name monitor element</td>
</tr>
<tr>
<td>CLIENT_WRKSTNNAME</td>
<td>VARCHAR(255)</td>
<td>client_wrkstnname - Client workstation name monitor element</td>
</tr>
<tr>
<td>CLIENT_ACCTNG</td>
<td>VARCHAR(255)</td>
<td>client_acctng - Client accounting string monitor element</td>
</tr>
<tr>
<td>CLIENT_USER</td>
<td>VARCHAR(255)</td>
<td>Current value of the CLIENT_USERID special register for this workload occurrence.</td>
</tr>
<tr>
<td>CLIENT_APPLNAME</td>
<td>VARCHAR(255)</td>
<td>client_applname - Client application name monitor element</td>
</tr>
<tr>
<td>COORD_ACT_COMPLETED_TOTAL</td>
<td>INTEGER</td>
<td>coord_act_completed_total - Coordinator activities completed total monitor element</td>
</tr>
<tr>
<td>COORD_ACT_ABORTED_TOTAL</td>
<td>INTEGER</td>
<td>coord_act_aborted_total - Coordinator activities aborted total monitor element</td>
</tr>
<tr>
<td>COORD_ACT_REJECTED_TOTAL</td>
<td>INTEGER</td>
<td>coord_act_rejected_total - Coordinator activities rejected total monitor element</td>
</tr>
</tbody>
</table>
Table 81. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCURRENT_ACT_TOP</td>
<td>INTEGER</td>
<td>concurrent_act_top - Concurrent activity top monitor element</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>VARCHAR(255)</td>
<td>address - IP address from which the connection was initiated</td>
</tr>
<tr>
<td>APPL_ID</td>
<td>VARCHAR(128)</td>
<td>appl_id - Application ID</td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member monitor element</td>
</tr>
<tr>
<td>COORD_MEMBER</td>
<td>SMALLINT</td>
<td>coord_member - Coordinator member monitor element</td>
</tr>
</tbody>
</table>

**MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses**

The MON_GET_SERVICE_SUBCLASS_STATS function returns basic statistics for one or more service subclasses.

**Authorization**

One of the following authorizations is required:
- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

**Default PUBLIC privilege**

None

**Syntax**

```plaintext
MON_GET_SERVICE_SUBCLASS_STATS (service_superclass_name, service_subclass_name, member)
```

The schema is SYSPROC.

**Routine parameters**

- `service_superclass_name`  
  An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, the data is retrieved for all of the superclasses in the database.

- `service_subclass_name`  
  An input argument of type VARCHAR(128) that specifies the name of a service subclass in the currently connected database. If the argument is null or an empty string, the data is retrieved for all of the subclasses in the database.

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

Information returned

Table 82. Information returned by the MON_GET_SERVICE_SUBCLASS_STATS routine

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_SUPERCLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>service_superclass_name - Service superclass name monitor element</td>
</tr>
<tr>
<td>SERVICE_SUBCLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>service_subclass_name - Service subclass name monitor element</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>LAST_RESET</td>
<td>TIMESTAMP</td>
<td>last_reset - Last Reset Timestamp monitor element</td>
</tr>
<tr>
<td>COORD_ACT_COMPLETED_TOTAL</td>
<td>BIGINT</td>
<td>coord_act_completed_total - Coordinator activities completed total monitor element</td>
</tr>
<tr>
<td>COORD_ACT_ABORTED_TOTAL</td>
<td>BIGINT</td>
<td>coord_act_aborted_total - Coordinator activities aborted total monitor element</td>
</tr>
<tr>
<td>COORD_ACT_REJECTED_TOTAL</td>
<td>BIGINT</td>
<td>coord_act_rejected_total - Coordinator activities rejected total monitor element</td>
</tr>
<tr>
<td>CONCURRENT_ACT_TOP</td>
<td>INTEGER</td>
<td>concurrent_act_top - Concurrent activity top monitor element</td>
</tr>
<tr>
<td>COORD_ACT_LIFETIME_TOP</td>
<td>BIGINT</td>
<td>coord_act_lifetime_top - Coordinator activity lifetime top monitor element</td>
</tr>
<tr>
<td>COORD_ACT_LIFETIME_AVG</td>
<td>DOUBLE</td>
<td>coord_act_lifetime_avg - Coordinator activity lifetime average monitor element</td>
</tr>
<tr>
<td>COORD_ACT_LIFETIME_STDDEV</td>
<td>DOUBLE</td>
<td>Standard deviation of lifetime for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds. This standard deviation is computed from the coordinator activity lifetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The COORD_ACT_LIFETIME_STDDEV value of a service subclass is unaffected by activities that pass through the service subclass but are remapped to a different subclass before they are completed.</td>
</tr>
<tr>
<td>COORD_ACT_EXEC_TIME_AVG</td>
<td>DOUBLE</td>
<td>coord_act_exec_time_avg - Coordinator activities execution time average monitor element</td>
</tr>
</tbody>
</table>
Table 82. Information returned by the MON_GET_SERVICE_SUBCLASS_STATS routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COORD_ACT_EXEC_TIME_STDDEV</td>
<td>DOUBLE</td>
<td>Standard deviation of the execution times for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. Units are milliseconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This standard deviation is computed from the coordinator activity execution time histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The execution time standard deviation of a service subclass is unaffected by activities that pass through the subclass but are remapped to a different subclass before they are completed.</td>
</tr>
<tr>
<td>COORD_ACT_QUEUE_TIME_AVG</td>
<td>DOUBLE</td>
<td>coord_act_queue_time_avg - Coordinator activity queue time average monitor element</td>
</tr>
<tr>
<td>COORD_ACT_QUEUE_TIME_STDDEV</td>
<td>DOUBLE</td>
<td>Standard deviation of the queue time for coordinator activities at nesting level 0 that were associated with this service subclass since the last reset. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null. Units are milliseconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This standard deviation is computed from the coordinator activity queue time histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The queue time standard deviation is counted only toward the service subclass in which the activity was queued.</td>
</tr>
<tr>
<td>NUM_REQUESTS_ACTIVE</td>
<td>BIGINT</td>
<td>Number of requests that are running in the service subclass at the time that this table function is running.</td>
</tr>
<tr>
<td>NUM_REQUESTS_TOTAL</td>
<td>BIGINT</td>
<td>Number of requests that finished running in this service subclass since the last reset. This finished state applies to any request regardless of its membership in an activity. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the service class is set to NONE, the value of the column is null.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The NUM_REQUESTS_TOTAL value of a service subclass is unaffected by requests that pass through the service subclass but are not completed in it.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REQUEST_EXEC_TIME_AVG</td>
<td>DOUBLE</td>
<td>request_exec_time_avg - Request execution time average monitor element</td>
</tr>
<tr>
<td>REQUEST_EXEC_TIME_STDDEV</td>
<td>DOUBLE</td>
<td>Standard deviation of the execution times for requests that were associated with this service subclass since the last reset. Units are milliseconds. If the COLLECT AGGREGATE REQUEST DATA parameter of the service class is set to NONE, the value of this column is NULL. This standard deviation is computed from the request execution time histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The execution time standard deviation of a service subclass is unaffected by requests that pass through the subclass but were not completed in it.</td>
</tr>
<tr>
<td>REQUEST_EXEC_TIME_TOTAL</td>
<td>BIGINT</td>
<td>Sum of the execution times for requests that were associated with this service subclass since the last reset. Units are milliseconds. If the COLLECT AGGREGATE REQUEST DATA parameter of the service class is set to NONE, the value of this column is NULL. This total is computed from the request execution time histogram and may be inaccurate if the histogram is not correctly sized to fit the data. The value of -1 is returned if any values fall into the last histogram bin. The execution time total of a service subclass is unaffected by requests that pass through the subclass but are not completed in it.</td>
</tr>
<tr>
<td>ACT_REMAPPED_IN</td>
<td>BIGINT</td>
<td>act_remapped_in - Activities remapped in monitor element</td>
</tr>
<tr>
<td>ACT_REMAPPED_OUT</td>
<td>BIGINT</td>
<td>act_remapped_out - Activities remapped out monitor element</td>
</tr>
<tr>
<td>CONCURRENT_WLO_TOP</td>
<td>INTEGER</td>
<td>concurrent_wlo_top - Concurrent workload occurrences top monitor element</td>
</tr>
<tr>
<td>UOW_TOTAL_TIME_TOP</td>
<td>BIGINT</td>
<td>uow_total_time_top - UOW total time top monitor element</td>
</tr>
<tr>
<td>UOW_THROUGHPUT</td>
<td>DOUBLE</td>
<td>uow_throughput - Unit of work throughput monitor element</td>
</tr>
<tr>
<td>UOW_LIFETIME_AVG</td>
<td>DOUBLE</td>
<td>uow_lifetime_avg - Unit of work lifetime average monitor element</td>
</tr>
</tbody>
</table>
Table 82. Information returned by the MON_GET_SERVICE_SUBCLASS_STATS routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UOW_COMPLETED_TOTAL</td>
<td>BIGINT</td>
<td>uow_completed_total - Total completed units of work monitor element</td>
</tr>
<tr>
<td>TOTAL_CPU_TIME</td>
<td>BIGINT</td>
<td>total_cpu_time - Total CPU time monitor element</td>
</tr>
<tr>
<td>TOTAL_DISP_RUN_QUEUE_TIME</td>
<td>BIGINT</td>
<td>total_disp_run_queue_time - Total dispatcher run queue time monitor element</td>
</tr>
<tr>
<td>ACT_THROUGHPUT</td>
<td>DOUBLE</td>
<td>act_throughput - Activity throughput monitor element</td>
</tr>
<tr>
<td>CPU_UTILIZATION</td>
<td>DOUBLE</td>
<td>cpu_utilization - CPU utilization monitor element</td>
</tr>
<tr>
<td>APP_ACT_COMPLETED_TOTAL</td>
<td>BIGINT</td>
<td>app_act_completed_total - Total successful external coordinator activities monitor element</td>
</tr>
<tr>
<td>APP_ACT_ABORTED_TOTAL</td>
<td>BIGINT</td>
<td>app_act_aborted_total - Total failed external coordinator activities monitor element</td>
</tr>
<tr>
<td>APP_ACT_REJECTED_TOTAL</td>
<td>BIGINT</td>
<td>app_act_rejected_total - Total rejected external coordinator activities monitor element</td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member monitor element</td>
</tr>
<tr>
<td>ACTIVE_HASH_GRPBYS_TOP</td>
<td>BIGINT</td>
<td>active_hash_grpby_top - Active hash Group By operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_HASH_JOINS_TOP</td>
<td>BIGINT</td>
<td>active_hash_joins_top - Active hash joins operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_Olap_FUNCS_TOP</td>
<td>BIGINT</td>
<td>active_olap_funcs_top - Active OLAP functions operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_PEAS_TOP</td>
<td>BIGINT</td>
<td>active_peas_top - Active partial early aggregations operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_PEDS_TOP</td>
<td>BIGINT</td>
<td>active_peds_top - Active partial early distinct operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_SORT_CONSUMERS_TOP</td>
<td>BIGINT</td>
<td>active_sort_consumers_top - Active sort consumers high watermark</td>
</tr>
<tr>
<td>ACTIVE_SORTS_TOP</td>
<td>BIGINT</td>
<td>active_sorts_top - Active Sorts high watermark</td>
</tr>
<tr>
<td>ACTIVE_COL_VECTOR_CONSUMERS_TOP</td>
<td>BIGINT</td>
<td>active_col_vector_consumers_top - Active columnar_vector consumers high watermark</td>
</tr>
<tr>
<td>SORT_CONSUMER_HEAP_TOP</td>
<td>BIGINT</td>
<td>sort_consumer_heap_top - Individual private sort heap consumer high watermark</td>
</tr>
<tr>
<td>SORT_CONSUMER_SHRHEAP_TOP</td>
<td>BIGINT</td>
<td>sort_consumer_shrheap_top - Individual shared sort heap consumer high watermark</td>
</tr>
<tr>
<td>SORT_HEAP_TOP</td>
<td>BIGINT</td>
<td>sort_heap_top - Sort private heap high watermark</td>
</tr>
<tr>
<td>SORT_SHRHEAP_TOP</td>
<td>BIGINT</td>
<td>sort_shrheap_top - Sort share heap high watermark</td>
</tr>
</tbody>
</table>
Usage notes

Some statistics are returned only if you set the COLLECT AGGREGATE ACTIVITY DATA and COLLECT AGGREGATE REQUEST DATA parameters for the corresponding service subclass to a value other than NONE.

The MON_GET_SERVICE_SUBCLASS_STATS table function returns one row of data per service subclass and per member. The function does not aggregate data across service classes (on a partition) or across partitions (for one or more service classes). However, you can use SQL queries to aggregate data.

The parameters are assessed in conjunction with each other. If you specify conflicting input parameters, no rows are returned. For example, if you specify a superclass named SUPA and a subclass named SUBB (where SUBB is not a subclass of SUPA), no rows are returned.

Examples

Example 1: Because every activity must be mapped to a DB2 service class before being run, you can monitor the global state of the system by using the service class statistics table functions and querying all of the service classes on all members. In the following example, a null value is passed for service superclass_name and service subclass_name to return statistics for all service classes, and the value -2 is specified for member to return statistics for all members:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       CAST(COORD_ACT_LIFETIME_AVG / 1000 AS DECIMAL(9,3))
       AS AVGLIFETIME,
       CAST(COORD_ACT_LIFETIME_STDDEV / 1000 AS DECIMAL(9,3))
       AS STDDEVLIFETIME,
       SUBSTR(CAST(LAST_RESET AS VARCHAR(30)),1,16) AS LAST_RESET
FROM TABLE(MON_GET_SERVICE_SUBCLASS_STATS(CAST(NULL AS VARCHAR(128)),
                                                  CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, MEMB
```

The statement returns service class statistics such as average activity lifetime and standard deviation in seconds, as shown in the following sample output:

```
SUPERCLASS_NAME SUBCLASS_NAME MEMB ... AVGLIFETIME STDDEVLIFETIME LAST_RESET
--------------- ---------------- ----- ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0 ... 691.242 34.322 2006-07-24-11.44
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1 ... 644.740 22.124 2006-07-24-11.44
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2 ... 612.431 43.347 2006-07-24-11.44
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3 ... 593.451 28.329 2006-07-24-11.44
... ...
```

Example 2: The same table function can also give the highest value for average concurrency of coordinator activities running in the service class on each member:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       CONCURRENT_ACT_TOP AS ACTTOP,
       CONCURRENT_WLO_TOP AS CONNTOP
FROM MON_GET_SERVICE_SUBCLASS_STATS(CAST(NULL AS VARCHAR(128)),
                                                  CAST(NULL AS VARCHAR(128)), -2)
```
FROM TABLE(MON_GET_SERVICE_SUBCLASS_STATS(CAST(NULL AS VARCHAR(128)),
        CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, MEMB

Sample output is as follows:

<table>
<thead>
<tr>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
<th>MEMB</th>
<th>ACTTOP</th>
<th>CONNTOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

By checking the average execution times and numbers of activities in the output of this table function, you can get a good high-level view of the load on each member for a specific database. Any significant variations in the high-level gauges returned by this table function might indicate a change in the load on the system.

Example 3: If an activity uses thresholds with REMAP ACTIVITY TO actions, the activity might spend time in more than one service class during its lifetime. You can determine how many activities have passed through a set of service classes by looking at the ACT_REMAPPED_IN and ACT_REMAPPED_OUT columns, as shown in the following example:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       ACT_REMAPPED_IN AS MAPPED_IN,
       ACT_REMAPPED_OUT AS MAPPED_OUT
FROM TABLE(MON_GET_SERVICE_SUBCLASS_STATS(CAST(NULL AS VARCHAR(128)),
                                          CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME
```

Sample output is as follows:

<table>
<thead>
<tr>
<th>SUPERCLASS_NAME</th>
<th>SUBCLASS_NAME</th>
<th>MAPPED_IN</th>
<th>MAPPED_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTUSERCLASS</td>
<td>SYSDEFAULTSUBCLASS</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SUPERCLASS1</td>
<td>SUBCLASS1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>SUPERCLASS1</td>
<td>SUBCLASS2</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

**MON_GET_SERVICE_SUPERCLASS_STATS - Return statistics of service superclasses**

The MON_GET_SERVICE_SUPERCLASS_STATS function returns basic statistics of one or more service superclasses.

**Authorization**

One of the following authorizations is required:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

**Default PUBLIC privilege**

None
Syntax

MON_GET_SERVICE_SUPERCLASS_STATS (service superclass name, member)

The schema is SYSPROC.

Routine parameters

service superclass name
An input argument of type VARCHAR(128) that specifies the name of a service superclass in the currently connected database. If the argument is null or an empty string, data is retrieved for all the superclasses in the database.

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

Information returned

Table 83. Information returned by the MON_GET_SERVICE_SUPERCLASS_STATS routine

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_SUPERCLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>service superclass name - Service superclass name monitor element</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>LAST_RESET</td>
<td>TIMESTAMP</td>
<td>last reset - Last Reset Timestamp monitor element</td>
</tr>
<tr>
<td>CONCURRENT_CONNECTION_TOP</td>
<td>INTEGER</td>
<td>concurrent_connection_top - Concurrent connection top monitor element</td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member monitor element</td>
</tr>
</tbody>
</table>

Usage notes

The MON_GET_SERVICE_SUPERCLASS_STATS table function returns one row of data per service superclass and per member. The function does not aggregate data across service superclasses (on a member) or across members (for one or more service superclasses). However, you can use SQL queries to aggregate data, as shown in the previous example.

Example

The following query displays the basic statistics for all the service superclasses on the system, across all database members:

```sql
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME, 1, 26) SERVICE_SUPERCLASS_NAME,
       MEMBER,
       LAST_RESET,
       CONCURRENT_CONNECTION_TOP CONCURRENT_CONN_TOP
FROM TABLE(MON_GET_SERVICE_SUPERCLASS_STATS('', -2)) as SCSTATS
```

Sample output is as follows:
MON_GET_WORK_ACTION_SET_STATS - Return work action set statistics

The MON_GET_WORK_ACTION_SET_STATS function returns the statistics for a work action set.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

Default PUBLIC privilege

None

Syntax

```sql
MON_GET_WORK_ACTION_SET_STATS (work_action_set_name, member)
```

The schema is SYSPROC.

Routine parameters

`work_action_set_name`

An input argument of type VARCHAR(128) that specifies the work action set to return statistics for. If the argument is null or an empty string, statistics are returned for all work action sets.

`member`

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

Information returned

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK_ACTION_SET_NAME</td>
<td>VARCHAR(128)</td>
<td>work_action_set_name - Work action set name monitor element</td>
</tr>
</tbody>
</table>
Table 84. Information returned for MON_GET_WORK_ACTION_SET_STATS (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>LAST_RESET</td>
<td>TIMESTAMP</td>
<td>last_reset - Last Reset Timestamp monitor element</td>
</tr>
<tr>
<td>WORK_CLASS_NAME</td>
<td>VARCHAR(128)</td>
<td>work_class_name - Work class name monitor element</td>
</tr>
<tr>
<td>ACT_TOTAL</td>
<td>BIGINT</td>
<td>act_total - Activities total monitor element</td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member monitor element</td>
</tr>
</tbody>
</table>

Example

Assume that there are three work classes: ReadClass, WriteClass, and LoadClass. There is a work action associated with ReadClass and a work action associated with LoadClass, but there is no work action associated with WriteClass. On member 0, the number of activities currently running or queued are as follows:

- ReadClass class: eight
- WriteClass class: four
- LoadClass class: two
- Unassigned: three

```sql
SELECT SUBSTR(WORK_ACTION_SET_NAME,1,18) AS WORK_ACTION_SET_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(WORK_CLASS_NAME,1,15) AS WORK_CLASS_NAME,
       LAST_RESET,
       SUBSTR(CHAR(ACT_TOTAL),1,14) AS ACT_TOTAL
FROM TABLE(MON_GET_WORK_ACTION_SET_STATS
           (CAST(NULL AS VARCHAR(128)), -2)) AS WASSTATS
ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME, MEMB
```

Sample output is as follows. Because there is no work action associated with the WriteClass work class, the four activities to which it applies are counted in the artificial class denoted by an asterisk (*) in the output. The three activities that were not assigned to any work class are also included in the artificial class.

<table>
<thead>
<tr>
<th>WORK_ACTION_SET_NAME</th>
<th>MEMB</th>
<th>WORK_CLASS_NAME</th>
<th>LAST_RESET</th>
<th>ACT_TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminActionSet</td>
<td>0</td>
<td>ReadClass</td>
<td>2005-11-25-18.52.49.343000</td>
<td>8</td>
</tr>
<tr>
<td>AdminActionSet</td>
<td>1</td>
<td>ReadClass</td>
<td>2005-11-25-18.52.50.478000</td>
<td>0</td>
</tr>
<tr>
<td>AdminActionSet</td>
<td>0</td>
<td>LoadClass</td>
<td>2005-11-25-18.52.49.343000</td>
<td>2</td>
</tr>
<tr>
<td>AdminActionSet</td>
<td>1</td>
<td>LoadClass</td>
<td>2005-11-25-18.52.50.478000</td>
<td>0</td>
</tr>
<tr>
<td>AdminActionSet</td>
<td>0</td>
<td>*</td>
<td>2005-11-25-18.52.49.343000</td>
<td>7</td>
</tr>
<tr>
<td>AdminActionSet</td>
<td>1</td>
<td>*</td>
<td>2005-11-25-18.52.50.478000</td>
<td>0</td>
</tr>
</tbody>
</table>

MON_GET_ACTIVITY table function - Return a list of activities

The MON_GET_ACTIVITY function returns the list of all activities that were submitted by the specified application on the specified member and have not yet been completed.

Refer to Table 85 on page 416 for a complete list of information that can be returned.

Authorization

One of the following authorizations is required:

- EXECUTE privilege on the routine
• DATAACCESS authority
• DBADM authority
• SQLADM authority
• WLMADM authority

**Default PUBLIC privilege**

None

**Syntax**

```
MON_GET_ACTIVITY(application_handle, member)
```

The schema is SYSPROC.

**Routine parameters**

*application_handle*

An input argument of type BIGINT that specifies an application handle for which a list of activities is to be returned. If the argument is null, the data is retrieved for all the applications in the database.

*member*

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

**Information returned**

*Table 85. Information returned by the MON_GET_ACTIVITY routine*

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_HANDLE</td>
<td>BIGINT</td>
<td>application_handle - Application handle monitor element</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>COORD_PARTITION_NUM</td>
<td>SMALLINT</td>
<td>coord_partition_num - Coordinator partition number monitor element</td>
</tr>
<tr>
<td>LOCAL_START_TIME</td>
<td>TIMESTAMP</td>
<td>local_start_time - Local start time monitor element</td>
</tr>
<tr>
<td>UOW_ID</td>
<td>INTEGER</td>
<td>uow_id - Unit of work ID monitor element</td>
</tr>
<tr>
<td>ACTIVITY_ID</td>
<td>INTEGER</td>
<td>activity_id - Activity ID monitor element</td>
</tr>
<tr>
<td>PARENT_UOW_ID</td>
<td>INTEGER</td>
<td>parent_uow_id - Parent unit of work ID monitor element</td>
</tr>
<tr>
<td>PARENT_ACTIVITY_ID</td>
<td>INTEGER</td>
<td>parent_activity_id - Parent activity ID monitor element</td>
</tr>
</tbody>
</table>
Table 85. Information returned by the MON_GET_ACTIVITY routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVITY_STATE</td>
<td>VARCHAR(32)</td>
<td>activity_state - Activity state monitor element Possible values are as follows: QUEUED: The activity is queued by a workload management queuing threshold. In a partitioned database environment, this state might mean that the coordinator agent has made an RPC to the catalog member to obtain threshold tickets and has not yet received a response. This state might indicate that the activity has been queued by a workload management queuing threshold or, if not much time has elapsed, can indicate that the activity is in the process of obtaining its tickets. To obtain a more accurate picture of whether the activity is being queued, determine what agent is working on the activity, and find out whether the EVENT_OBJECT value of the object at the catalog member has a value of WLM_QUEUE. TERMINATING: The activity has finished running and is being removed from the system.</td>
</tr>
<tr>
<td>ACTIVITY_TYPE</td>
<td>VARCHAR(32)</td>
<td>Activity type. Possible values are as follows: CALL, DDL, LOAD, OTHER, READ_DML, WRITE_DML</td>
</tr>
<tr>
<td>NESTING_LEVEL</td>
<td>INTEGER</td>
<td>nesting_level - Nesting level monitor element</td>
</tr>
<tr>
<td>INVOCATION_ID</td>
<td>INTEGER</td>
<td>invocation_id - Invocation ID monitor element</td>
</tr>
<tr>
<td>ROUTINE_ID</td>
<td>INTEGER</td>
<td>routine_id - Routine ID monitor element</td>
</tr>
<tr>
<td>UTILITY_ID</td>
<td>INTEGER</td>
<td>utility_id - Utility ID monitor element</td>
</tr>
<tr>
<td>SERVICE_CLASS_ID</td>
<td>INTEGER</td>
<td>service_class_id - Service class ID monitor element</td>
</tr>
<tr>
<td>DATABASE_WORK_ACTION_SET_ID</td>
<td>INTEGER</td>
<td>One of the following values:  • If this activity has been categorized into a work class of database scope, the value is the ID of the work class set of which this work class is a member.  • If this activity has not been categorized into a work class of database scope, the value is null.</td>
</tr>
<tr>
<td>DATABASE_WORK_CLASS_ID</td>
<td>INTEGER</td>
<td>One of the following values:  • If this activity has been categorized into a work class of database scope, the value is the ID of the work class.  • If this activity has not been categorized into a work class of database scope, the value is null.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| SERVICE_CLASS_WORK_ACTION_SET_ID | INTEGER | One of the following values:  
  • If this activity has been categorized into a work class of service class scope, the value is the ID of the work action set associated with the work class set to which the work class belongs.  
  • If this activity has not been categorized into a work class of service class scope, the value is null. |
| SERVICE_CLASS_WORK_CLASS_ID | INTEGER | One of the following values:  
  • If this activity has been categorized into a work class of service class scope, the value is the ID of the work class assigned to this activity.  
  • If this activity has not been categorized into a work class of service class scope, the value is null. |
| EXECUTABLE_ID | VARCHAR(32) FOR BIT DATA | executable_id - Executable ID monitor element |
| TOTAL_CPU_TIME | BIGINT | total_cpu_time - Total CPU time |
| ROWS_READ | BIGINT | rows_read - Rows read |
| ROWS_RETURNED | BIGINT | rows_returned - Rows returned |
| QUERY_COST_ESTIMATE | BIGINT | query_cost_estimate - Query cost estimate |
| DIRECT_READS | BIGINT | direct_reads - Direct reads from database |
| DIRECT_WRITES | BIGINT | direct_writes - Direct writes to database |
| ENTRY_TIME | TIMESTAMP | entry_time - Entry time |
| MEMBER | SMALLINT | member - Database member monitor element |
| COORD_MEMBER | SMALLINT | coord_member - Coordinator member monitor element |
| PACKAGE_SCHEMA | VARCHAR(128) | package_schema - Package schema |
| PACKAGE_NAME | VARCHAR(128) | package_name - Package name |
| PACKAGE_VERSION_ID | VARCHAR(64) | package_version_id - Package version |
| SECTION_NUMBER | BIGINT | section_number - Section number |
| STMNTNO | INTEGER | stmtno - Statement number monitor element |
| APPL_ID | VARCHAR(128) | appl_id - Application ID monitor element |
| CLIENT_ACCTNG | VARCHAR(255) | client_acctng - Client accounting string monitor element |
| CLIENT_APPLNAME | VARCHAR(255) | client_applname - Client application name monitor element |
| CLIENT_USERID | VARCHAR(255) | client_userid - Client user ID monitor element |
| CLIENT_WRKSTNNNAME | VARCHAR(255) | client_wrkstnnname - Client workstation name monitor element |
| EFFECTIVE_ISOLATION | CHAR(2) | effective_isolation - Effective isolation monitor element |
| EFFECTIVE_LOCK_TIMEOUT | BIGINT | effective_lock_timeout - Effective lock timeout monitor element |
| EFFECTIVE_QUERY_DEGREE | BIGINT | effective_query_degree - Effective query degree monitor element |
| ACTIVITYTOTALTIME_THRESHOLD_ID | INTEGER | activitytotaltime_threshold_id - Activity total time threshold ID monitor element |
| ACTIVITYTOTALTIME_THRESHOLD_VALUE | BIGINT | activitytotaltime_threshold_value - Activity total time threshold value monitor element |
Table 85. Information returned by the MON_GET_ACTIVITY routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVITYTOTALTIME_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>activitytotaltime_threshold_violated - Activity total time threshold violated monitor element</td>
</tr>
<tr>
<td>AGGSQLTEMPSPACE_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>aggsqltempspace_threshold_id - Aggregate SQL temporary space threshold ID monitor element</td>
</tr>
<tr>
<td>AGGSQLTEMPSPACE_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>aggsqltempspace_threshold_value - AggSQL temporary space threshold value monitor element</td>
</tr>
<tr>
<td>AGGSQLTEMPSPACE_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>aggsqltempspace_threshold_violated - AggSQL temporary space threshold violated monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>concurrentdbcoordactivities_db_threshold_id - Concurrent database coordinator activities database threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_QUEUED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_db_threshold_queued - Concurrent database coordinator activities database threshold queued monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>concurrentdbcoordactivities_db_threshold_value - Concurrent database coordinator activities database threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_db_threshold_violated - Concurrent database coordinator activities database threshold violated monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>concurrentdbcoordactivities_subclass_threshold_id - Concurrent database coordinator activities service subclass threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_QUEUED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_subclass_thresholdqueued - Concurrent database coordinator activities service subclass threshold queued monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>concurrentdbcoordactivities_subclass_threshold_value - Concurrent database coordinator activities service subclass threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUBCLASS_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_subclass_threshold_violated - Concurrent database coordinator activities service subclass threshold violated monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>concurrentdbcoordactivities superclass_threshold_id - Concurrent database coordinator activities service superclass threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_QUEUED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_s superclass_threshold_queued - Concurrent database coordinator activities service superclass threshold queued monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>concurrentdbcoordactivities_s superclass_threshold_value - Concurrent database coordinator activities service superclass threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_SUPERCLASS_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_s superclass_threshold_violated - Concurrent database coordinator activities service superclass threshold violated monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WL_WAS_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>concurrentdbcoordactivities_wl_was_threshold_id - Concurrent database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coordinator activities workload work action set threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WL_WAS_THRESHOLD_QUEUED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_wl_was_threshold_queued - Concurrent database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coordinator activities workload work action set threshold queued monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WL_WAS_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>concurrentdbcoordactivities_wl_was_threshold_value - Concurrent database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coordinator activities workload work action set threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WL_WAS_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_wl_was_threshold_violated - Concurrent database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coordinator activities workload work action set threshold violated monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_id - Concurrent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>database coordinator activities work action set threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_QUEUED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_queued - Concurrent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>database coordinator activities work action set threshold queued monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_value - Concurrent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>database coordinator activities work action set threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>concurrentdbcoordactivities_work_action_set_threshold_violated - Concurrent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>database coordinator activities work action set threshold violated monitor element</td>
</tr>
<tr>
<td>CONCURRENTWORKLOADACTIVITIES_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>concurrentworkloadactivities_threshold_id - Concurrent workload activities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>threshold ID monitor element</td>
</tr>
<tr>
<td>CONCURRENTWORKLOADACTIVITIES_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>concurrentworkloadactivities_threshold_value - Concurrent workload activities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>threshold value monitor element</td>
</tr>
<tr>
<td>CONCURRENTWORKLOADACTIVITIES_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>concurrentworkloadactivities_threshold_violated - Concurrent workload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>activities threshold violated monitor element</td>
</tr>
<tr>
<td>CPUS_TIME_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>cputime_threshold_id - CPU time threshold ID monitor element</td>
</tr>
<tr>
<td>CPUS_TIME_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>cputime_threshold_value - CPU time threshold value monitor element</td>
</tr>
<tr>
<td>CPUS_TIME_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>cputime_threshold_violated - CPU time threshold violated monitor element</td>
</tr>
<tr>
<td>CPUS_TIMEINSC_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>cputimeinsc_threshold_id - CPU time in service class threshold ID monitor element</td>
</tr>
<tr>
<td>CPUS_TIMEINSC_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>cputimeinsc_threshold_value - CPU time in service class threshold value monitor element</td>
</tr>
<tr>
<td>CPUS_TIMEINSC_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>cputimeinsc_threshold_violated - CPU time in service class threshold violated monitor element</td>
</tr>
<tr>
<td>DATATAGINSC_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>datataginsc_threshold_id - Data tag in service class threshold (IN condition) ID monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DATATAGINSC_THRESHOLD_VALUE</td>
<td>VARCHAR(32)</td>
<td>datataginsc_threshold_value - Data tag in service class threshold (IN condition) value monitor element</td>
</tr>
<tr>
<td>DATATAGINSC_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>datataginsc_threshold_violated - Data tag in service class threshold (IN condition) violated monitor element</td>
</tr>
<tr>
<td>DATATAGNOTINSC_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>datatagnotinsc_threshold_id - Data tag in service class threshold (NOT IN condition) ID monitor element</td>
</tr>
<tr>
<td>DATATAGNOTINSC_THRESHOLD_VALUE</td>
<td>VARCHAR(32)</td>
<td>datatagnotinsc_threshold_value - Data tag in service class threshold (NOT IN condition) value monitor element</td>
</tr>
<tr>
<td>DATATAGNOTINSC_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>datatagnotinsc_threshold_violated - Data tag in service class threshold (NOT IN condition) violated monitor element</td>
</tr>
<tr>
<td>ESTIMATEDSQLCOST_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>estimatedsqlcost_threshold_id - Estimated SQL cost threshold ID monitor element</td>
</tr>
<tr>
<td>ESTIMATEDSQLCOST_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>estimatedsqlcost_threshold_value - Estimated SQL cost threshold value monitor element</td>
</tr>
<tr>
<td>ESTIMATEDSQLCOST_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>estimatedsqlcost_threshold_violated - Estimated SQL cost threshold violated monitor element</td>
</tr>
<tr>
<td>LAST_REFERENCE_TIME</td>
<td>TIMESTAMP</td>
<td>last_reference_time - Last reference time monitor element</td>
</tr>
<tr>
<td>QUERY_DATA_TAG_LIST</td>
<td>VARCHAR(32)</td>
<td>query_data_tag_list - Estimated query data tag list monitor element</td>
</tr>
<tr>
<td>QUERY_ACTUAL_DEGREE</td>
<td>INTEGER</td>
<td>query_actual_degree - Actual runtime degree of intrapartition parallelism monitor element</td>
</tr>
<tr>
<td>SQLROWSREAD_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>sqlrowsread_threshold_id - SQL rows read threshold ID monitor element</td>
</tr>
<tr>
<td>SQLROWSREAD_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>sqlrowsread_threshold_value - SQL rows read threshold value monitor element</td>
</tr>
<tr>
<td>SQLROWSREAD_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>sqlrowsread_threshold_violated - SQL rows read threshold violated monitor element</td>
</tr>
<tr>
<td>SQLROWSREADINSC_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>sqlrowsreadinsc_threshold_id - SQL rows read in service class threshold ID monitor element</td>
</tr>
<tr>
<td>SQLROWSREADINSC_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>sqlrowsreadinsc_threshold_value - SQL rows read in service class threshold value monitor element</td>
</tr>
<tr>
<td>SQLROWSREADINSC_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>sqlrowsreadinsc_threshold_violated - SQL rows read in service class threshold violated monitor element</td>
</tr>
<tr>
<td>SQLROWSRETURNED_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>sqlrowsreturned_threshold_id - SQL rows read returned threshold ID monitor element</td>
</tr>
<tr>
<td>SQLROWSRETURNED_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>sqlrowsreturned_threshold_value - SQL rows read returned threshold value monitor element</td>
</tr>
<tr>
<td>SQLROWSRETURNED_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>sqlrowsreturned_threshold_violated - SQL rows read returned threshold violated monitor element</td>
</tr>
<tr>
<td>SQLTEMPSPACE_THRESHOLD_ID</td>
<td>INTEGER</td>
<td>sqltempspace_threshold_id - SQL temporary space threshold ID monitor element</td>
</tr>
<tr>
<td>SQLTEMPSPACE_THRESHOLD_VALUE</td>
<td>BIGINT</td>
<td>sqltempspace_threshold_value - SQL temporary space threshold value monitor element</td>
</tr>
<tr>
<td>SQLTEMPSPACE_THRESHOLD_VIOLATED</td>
<td>SMALLINT</td>
<td>sqltempspace_threshold_violated - SQL temporary space threshold violated monitor element</td>
</tr>
<tr>
<td>WL_WORK_ACTION_SET_ID</td>
<td>INTEGER</td>
<td>wl_work_action_set_id - Workload work action set identifier monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WL_WORK_CLASS_ID</td>
<td>INTEGER</td>
<td>wl_work_class_id - Workload work class identifier monitor element</td>
</tr>
<tr>
<td>STMT_TEXT</td>
<td>CLOB(2M)</td>
<td>stmt_text - SQL statement text monitor element</td>
</tr>
<tr>
<td>EFF_STMT_TEXT</td>
<td>CLOB(2M)</td>
<td>eff_stmt_text - Effective statement text monitor element</td>
</tr>
<tr>
<td>IS_SYSTEM_APPL</td>
<td>SMALLINT</td>
<td>is_system_appl - Is System Application monitor element</td>
</tr>
<tr>
<td>NUM_AGENTS</td>
<td>BIGINT</td>
<td>num_agents - Number of Agents Working on a Statement monitor element</td>
</tr>
<tr>
<td>AGENTS_TOP</td>
<td>BIGINT</td>
<td>agents_top - Number of Agents Created monitor element</td>
</tr>
<tr>
<td>QUERY_CARD_ESTIMATE</td>
<td>BIGINT</td>
<td>query_card_estimate - Query Number of Rows Estimate monitor element</td>
</tr>
<tr>
<td>UTILITY_INVOCATION_ID</td>
<td>VARCHAR(32) FOR BIT DATA</td>
<td>utility_invocation_id - Utility invocation ID monitor element</td>
</tr>
<tr>
<td>STMT_PKG_CACHE_ID</td>
<td>BIGINT</td>
<td>stmt_pkgcache_id - Statement package cache identifier monitor element</td>
</tr>
<tr>
<td>INTRA_PARALLEL_STATE</td>
<td>VARCHAR(3)</td>
<td>intra_parallel_state - Current state of intrapartition parallelism monitor element</td>
</tr>
<tr>
<td>NUM_ROUTINES</td>
<td>BIGINT</td>
<td>num_routines - Number of routines monitor element</td>
</tr>
<tr>
<td>TOTAL_ACT_TIME</td>
<td>BIGINT</td>
<td>total_act_time - Total activity time monitor element</td>
</tr>
<tr>
<td>TOTAL_ACT_WAIT_TIME</td>
<td>BIGINT</td>
<td>total_act_wait_time - Total activity wait time monitor element</td>
</tr>
<tr>
<td>POOL_READ_TIME</td>
<td>BIGINT</td>
<td>pool_read_time - Total buffer pool physical read time monitor element</td>
</tr>
<tr>
<td>POOL_WRITE_TIME</td>
<td>BIGINT</td>
<td>pool_write_time - Total buffer pool physical write time monitor element</td>
</tr>
<tr>
<td>DIRECT_READ_TIME</td>
<td>BIGINT</td>
<td>direct_read_time - Direct read time monitor element</td>
</tr>
<tr>
<td>DIRECT_WRITE_TIME</td>
<td>BIGINT</td>
<td>direct_write_time - Direct write time monitor element</td>
</tr>
<tr>
<td>LOCK_WAIT_TIME</td>
<td>BIGINT</td>
<td>lock_wait_time - Time waited on locks monitor element</td>
</tr>
<tr>
<td>TOTAL_SECTION_SORT_TIME</td>
<td>BIGINT</td>
<td>total_section_sort_time - Total section sort time monitor element</td>
</tr>
<tr>
<td>TOTAL_SECTION_SORT_PROC_TIME</td>
<td>BIGINT</td>
<td>total_section_sort_proc_time - Total section sort processing time monitor element</td>
</tr>
<tr>
<td>TOTAL_SECTION_SORTS</td>
<td>BIGINT</td>
<td>total_section_sorts - Total section sorts monitor element</td>
</tr>
<tr>
<td>LOCK_ESCALS</td>
<td>BIGINT</td>
<td>lock_escals - Number of lock escalations monitor element</td>
</tr>
<tr>
<td>LOCK_WAITS</td>
<td>BIGINT</td>
<td>lock_waits - Lock waits monitor element</td>
</tr>
<tr>
<td>ROWS_MODIFIED</td>
<td>BIGINT</td>
<td>rows_modified - Rows modified monitor element</td>
</tr>
<tr>
<td>DIRECT_READ_REQS</td>
<td>BIGINT</td>
<td>direct_read_reqs - Direct read requests monitor element</td>
</tr>
<tr>
<td>DIRECT_WRITE_REQS</td>
<td>BIGINT</td>
<td>direct_write_reqs - Direct write requests monitor element</td>
</tr>
<tr>
<td>POOL_DATA_L_READS</td>
<td>BIGINT</td>
<td>pool_data_l_reads - Buffer pool data logical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_DATA_L_READS</td>
<td>BIGINT</td>
<td>pool_temp_data_l_reads - Buffer pool temporary data logical reads monitor element</td>
</tr>
</tbody>
</table>
Table 85. Information returned by the MON_GET_ACTIVITY routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POOL_XDA_L_READS</td>
<td>BIGINT</td>
<td>pool_xda_l_reads - Buffer pool XDA data logical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_XDA_L_READS</td>
<td>BIGINT</td>
<td>pool_temp_xda_l_reads - Buffer pool temporary XDA data logical reads monitor element</td>
</tr>
<tr>
<td>POOL_INDEX_L_READS</td>
<td>BIGINT</td>
<td>pool_index_l_reads - Buffer pool index logical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_INDEX_L_READS</td>
<td>BIGINT</td>
<td>pool_temp_index_l_reads - Buffer pool temporary index logical reads monitor element</td>
</tr>
<tr>
<td>POOL_DATA_P_READS</td>
<td>BIGINT</td>
<td>pool_data_p_reads - Buffer pool data physical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_DATA_P_READS</td>
<td>BIGINT</td>
<td>pool_temp_data_p_reads - Buffer pool temporary data physical reads monitor element</td>
</tr>
<tr>
<td>POOL_XDA_P_READS</td>
<td>BIGINT</td>
<td>pool_xda_p_reads - Buffer pool temporary data physical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_XDA_P_READS</td>
<td>BIGINT</td>
<td>pool_temp_xda_p_reads - Buffer pool temporary XDA data physical reads monitor element</td>
</tr>
<tr>
<td>POOL_INDEX_P_READS</td>
<td>BIGINT</td>
<td>pool_index_p_reads - Buffer pool index physical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_INDEX_P_READS</td>
<td>BIGINT</td>
<td>pool_temp_index_p_reads - Buffer pool temporary index physical reads monitor element</td>
</tr>
<tr>
<td>POOL_DATA_WRITES</td>
<td>BIGINT</td>
<td>pool_data_writes - Buffer pool data writes monitor element</td>
</tr>
<tr>
<td>POOL_XDA_WRITES</td>
<td>BIGINT</td>
<td>pool_xda_writes - Buffer pool XDA data writes monitor element</td>
</tr>
<tr>
<td>POOL_INDEX_WRITES</td>
<td>BIGINT</td>
<td>pool_index_writes - Buffer pool index writes monitor element</td>
</tr>
<tr>
<td>TOTAL_SORTS</td>
<td>BIGINT</td>
<td>total_sorts - Total sorts monitor element</td>
</tr>
<tr>
<td>POST_THRESHOLD_SORTS</td>
<td>BIGINT</td>
<td>post_threshold_sorts - Post threshold sorts monitor element</td>
</tr>
<tr>
<td>POST_SHR_THRESHOLD_SORTS</td>
<td>BIGINT</td>
<td>post_shrthreshold_sorts - Post shared threshold sorts monitor element</td>
</tr>
<tr>
<td>SORT_OVERFLOWS</td>
<td>BIGINT</td>
<td>sort_overflows - Sort overflows monitor element</td>
</tr>
<tr>
<td>WLM_QUEUE_TIME_TOTAL</td>
<td>BIGINT</td>
<td>wlm_queue_time_total - Workload manager total queue time monitor element</td>
</tr>
<tr>
<td>WLM_QUEUE_ASSIGNMENTS_TOTAL</td>
<td>BIGINT</td>
<td>wlm_queue_assignments_total - Workload manager total queue assignments monitor element</td>
</tr>
<tr>
<td>DEADLOCKS</td>
<td>BIGINT</td>
<td>deadlocks - Deadlocks detected monitor element</td>
</tr>
<tr>
<td>FCM_RECV_VOLUME</td>
<td>BIGINT</td>
<td>fcm_recv_volume - FCM received volume monitor element</td>
</tr>
<tr>
<td>FCM_RECVS_TOTAL</td>
<td>BIGINT</td>
<td>fcm_recvs_total - FCM receives total monitor element</td>
</tr>
<tr>
<td>FCM_SEND_VOLUME</td>
<td>BIGINT</td>
<td>fcm_send_volume - FCM send volume monitor element</td>
</tr>
<tr>
<td>FCM_SENDS_TOTAL</td>
<td>BIGINT</td>
<td>fcm_sends_total - FCM sends total monitor element</td>
</tr>
<tr>
<td>FCM_RECV_WAIT_TIME</td>
<td>BIGINT</td>
<td>fcm_recv_wait_time - FCM received wait time monitor element</td>
</tr>
<tr>
<td>FCM_SEND_WAIT_TIME</td>
<td>BIGINT</td>
<td>fcm_send_wait_time - FCM send wait time monitor element</td>
</tr>
<tr>
<td>LOCK_TIMEOUTS</td>
<td>BIGINT</td>
<td>lock_timeouts - Number of lock timeouts monitor element</td>
</tr>
<tr>
<td>LOG_BUFFER_WAIT_TIME</td>
<td>BIGINT</td>
<td>log_buffer_wait_time - Log buffer wait time monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUM_LOG_BUFFER_FULL</td>
<td>BIGINT</td>
<td>num_log_buffer_full - Number of times full log buffer caused agents to wait monitor element</td>
</tr>
<tr>
<td>LOG_DISK_WAIT_TIME</td>
<td>BIGINT</td>
<td>log_disk_wait_time - Log disk wait time monitor element</td>
</tr>
<tr>
<td>LOG_DISK_WAITS_TOTAL</td>
<td>BIGINT</td>
<td>log_disk_waits_total - Total log disk waits monitor element</td>
</tr>
<tr>
<td>TOTAL_ROUTINE_TIME</td>
<td>BIGINT</td>
<td>total_routine_time - Total routine time monitor element</td>
</tr>
<tr>
<td>TOTAL_ROUTINE_INVOCATIONS</td>
<td>BIGINT</td>
<td>total_routine_invocations - Total routine invocations monitor elements</td>
</tr>
<tr>
<td>COORD_STMT_EXEC_TIME</td>
<td>BIGINT</td>
<td>coord_stmt_exec_time - Execution time for statement by coordinator agent monitor element</td>
</tr>
<tr>
<td>STMT_EXEC_TIME</td>
<td>BIGINT</td>
<td>stmt_exec_time - Statement execution time monitor element</td>
</tr>
<tr>
<td>TOTAL_SECTION_TIME</td>
<td>BIGINT</td>
<td>total_section_time - Total section time monitor element</td>
</tr>
<tr>
<td>TOTAL_SECTION_PROC_TIME</td>
<td>BIGINT</td>
<td>total_section_proc_time - Total section processing time monitor element</td>
</tr>
<tr>
<td>TOTAL_ROUTINE_NON_SECT_TIME</td>
<td>BIGINT</td>
<td>total_routine_non_sect_time - Non-section routine execution time monitor elements</td>
</tr>
<tr>
<td>TOTAL_ROUTINE_NON_SECT_PROC_TIME</td>
<td>BIGINT</td>
<td>total_routine_non_sect_proc_time - Non-section processing time monitor element</td>
</tr>
<tr>
<td>LOCK_WAITS_GLOBAL</td>
<td>BIGINT</td>
<td>lock_waits_global - Lock waits global monitor element</td>
</tr>
<tr>
<td>LOCK_WAIT_TIME_GLOBAL</td>
<td>BIGINT</td>
<td>lock_wait_time_global - Lock wait time global monitor element</td>
</tr>
<tr>
<td>LOCK_TIMEOUTS_GLOBAL</td>
<td>BIGINT</td>
<td>lock_timeouts_global - Lock timeouts global monitor element</td>
</tr>
<tr>
<td>LOCK_ESCALS_MAXLOCKS</td>
<td>BIGINT</td>
<td>lock_escals_maxlocks - Number of maxlocks lock escalations monitor element</td>
</tr>
<tr>
<td>LOCK_ESCALS_LOCKLIST</td>
<td>BIGINT</td>
<td>lock_escals_locklist - Number of locklist lock escalations monitor element</td>
</tr>
<tr>
<td>LOCK_ESCALS_GLOBAL</td>
<td>BIGINT</td>
<td>lock_escals_global - Number of global lock escalations monitor element</td>
</tr>
<tr>
<td>RECLAIM_WAIT_TIME</td>
<td>BIGINT</td>
<td>reclaim_wait_time - Reclaim wait time monitor element</td>
</tr>
<tr>
<td>SPACEMAPPAGE_RECLAIM_WAIT_TIME</td>
<td>BIGINT</td>
<td>spacemappage_reclaim_wait_time - Space map page reclaim wait time monitor element</td>
</tr>
<tr>
<td>CF_WAITS</td>
<td>BIGINT</td>
<td>cf_waits - Number of cluster caching facility waits monitor element</td>
</tr>
<tr>
<td>CF_WAIT_TIME</td>
<td>BIGINT</td>
<td>cf_wait_time - cluster caching facility wait time monitor element</td>
</tr>
<tr>
<td>POOL_DATA_GBP_L_READS</td>
<td>BIGINT</td>
<td>pool_data_gbp_l_reads - Group buffer pool data logical reads monitor element</td>
</tr>
<tr>
<td>POOL_DATA_GBP_P_READS</td>
<td>BIGINT</td>
<td>pool_data_gbp_p_reads - Group buffer pool data physical reads monitor element</td>
</tr>
<tr>
<td>POOL_DATA_LBP_PAGES_FOUND</td>
<td>BIGINT</td>
<td>pool_data_lbp_pages_found - Local buffer pool found data pages monitor element</td>
</tr>
<tr>
<td>POOL_DATA_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>pool_data_gbp_invalid_pages - Group buffer pool invalid data pages monitor element</td>
</tr>
<tr>
<td>POOL_INDEX_GBP_L_READS</td>
<td>BIGINT</td>
<td>pool_index_gbp_l_reads - Group buffer pool index logical reads monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>POOL_INDEX_GBP_P_READS</td>
<td>BIGINT</td>
<td>pool_index_gbp_p_reads - Group buffer pool index physical reads monitor elements</td>
</tr>
<tr>
<td>POOL_INDEX_LBP_PAGES_FOUND</td>
<td>BIGINT</td>
<td>pool_index_lbp_pages_found - Local buffer pool index pages found monitor element</td>
</tr>
<tr>
<td>POOL_INDEX_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>pool_index_gbp_invalid_pages - Group buffer pool invalid index pages monitor element</td>
</tr>
<tr>
<td>POOL_XDA_GBP_L_READS</td>
<td>BIGINT</td>
<td>pool_xda_gbp_l_reads - Group buffer pool XDA data logical read requests monitor element</td>
</tr>
<tr>
<td>POOL_XDA_GBP_P_READS</td>
<td>BIGINT</td>
<td>pool_xda_gbp_p_reads - Group buffer pool XDA data physical read requests monitor element</td>
</tr>
<tr>
<td>POOL_XDA_LBP_PAGES_FOUND</td>
<td>BIGINT</td>
<td>pool_xda_lbp_pages_found - Local buffer pool XDA data pages found monitor element</td>
</tr>
<tr>
<td>POOL_XDA_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>pool_xda_gbp_invalid_pages - Group buffer pool invalid XDA data pages monitor element</td>
</tr>
<tr>
<td>AUDIT_EVENTS_TOTAL</td>
<td>BIGINT</td>
<td>audit_events_total - Total audit events monitor element</td>
</tr>
<tr>
<td>AUDIT_FILE_WRITES_TOTAL</td>
<td>BIGINT</td>
<td>audit_file_writes_total - Total audit files written monitor element</td>
</tr>
<tr>
<td>AUDIT_FILE_WRITE_WAIT_TIME</td>
<td>BIGINT</td>
<td>audit_file_write_wait_time - Audit file write wait time monitor element</td>
</tr>
<tr>
<td>AUDIT_SUBSYSTEM_WAITS_TOTAL</td>
<td>BIGINT</td>
<td>audit_subsystem_waits_total - Total audit subsystem waits monitor element</td>
</tr>
<tr>
<td>AUDIT_SUBSYSTEM_WAIT_TIME</td>
<td>BIGINT</td>
<td>audit_subsystem_wait_time - Audit subsystem wait time monitor element</td>
</tr>
<tr>
<td>DIAGLOG_WRITES_TOTAL</td>
<td>BIGINT</td>
<td>diaglog_writes_total - Total diagnostic log file writes monitor element</td>
</tr>
<tr>
<td>DIAGLOG_WRITE_WAIT_TIME</td>
<td>BIGINT</td>
<td>diaglog_write_wait_time - Diagnostic log file write wait time monitor element</td>
</tr>
<tr>
<td>FCM_MESSAGE_RECVS_TOTAL</td>
<td>BIGINT</td>
<td>fcm_message_recv_total - Total FCM message receives monitor element</td>
</tr>
<tr>
<td>FCM_MESSAGE_RECV_VOLUME</td>
<td>BIGINT</td>
<td>fcm_message_recv_volume - FCM message received volume monitor element</td>
</tr>
<tr>
<td>FCM_MESSAGE_RECV_WAIT_TIME</td>
<td>BIGINT</td>
<td>fcm_message_recv_wait_time - FCM message received wait time monitor element</td>
</tr>
<tr>
<td>FCM_MESSAGE SENDS_TOTAL</td>
<td>BIGINT</td>
<td>fcm_message_sends_total - Total FCM message sends monitor element</td>
</tr>
<tr>
<td>FCM_MESSAGE_SEND_VOLUME</td>
<td>BIGINT</td>
<td>fcm_message_send_volume - FCM message send volume monitor element</td>
</tr>
<tr>
<td>FCM_MESSAGE_SEND_WAIT_TIME</td>
<td>BIGINT</td>
<td>fcm_message_send_wait_time - FCM message send wait time monitor element</td>
</tr>
<tr>
<td>FCM_TQ_RECVS_TOTAL</td>
<td>BIGINT</td>
<td>fcm_tq_recv_total - FCM table queue receives total monitor element</td>
</tr>
<tr>
<td>FCM_TQ_RECV_VOLUME</td>
<td>BIGINT</td>
<td>fcm_tq_recv_volume - FCM table queue received volume monitor element</td>
</tr>
<tr>
<td>FCM_TQ_RECV_WAIT_TIME</td>
<td>BIGINT</td>
<td>fcm_tq_recv_wait_time - FCM table queue received wait time monitor element</td>
</tr>
<tr>
<td>FCM_TQ SENDS_TOTAL</td>
<td>BIGINT</td>
<td>fcm_tq_sends_total - FCM table queue send total monitor element</td>
</tr>
<tr>
<td>FCM_TQ_SEND_VOLUME</td>
<td>BIGINT</td>
<td>fcm_tq_send_volume - FCM table queue send volume monitor element</td>
</tr>
<tr>
<td>FCM_TQ_SEND_WAIT_TIME</td>
<td>BIGINT</td>
<td>fcm_tq_send_wait_time - FCM table queue send wait time monitor element</td>
</tr>
</tbody>
</table>
Table 85. Information returned by the MON_GET_ACTIVITY routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_LW_THRESH_EXCEEDED</td>
<td>BIGINT</td>
<td>num_lw_thresh_exceeded - Number of lock wait thresholds exceeded monitor element</td>
</tr>
<tr>
<td>THRESH_VIOLATIONS</td>
<td>BIGINT</td>
<td>thresh_violations - Number of threshold violations monitor element</td>
</tr>
<tr>
<td>TOTAL_APP_SECTION_EXECUTIONS</td>
<td>BIGINT</td>
<td>total_app_section_executions - Total application section executions monitor element</td>
</tr>
<tr>
<td>TOTAL_ROUTINE_USER_CODE_PROC_TIME</td>
<td>BIGINT</td>
<td>total_routine_user_code_proc_time - Total routine user code processing time monitor element</td>
</tr>
<tr>
<td>TOTAL_ROUTINE_USER_CODE_TIME</td>
<td>BIGINT</td>
<td>total_routine_user_code_time - Total routine user code time monitor element</td>
</tr>
<tr>
<td>TQ_TOT_SEND_SPILLS</td>
<td>BIGINT</td>
<td>tq_tot_send_spills - Total number of table queue buffers overflowed monitor element</td>
</tr>
<tr>
<td>EVMON_WAIT_TIME</td>
<td>BIGINT</td>
<td>evmon_wait_time - Event monitor wait time monitor element</td>
</tr>
<tr>
<td>EVMON_WAITS_TOTAL</td>
<td>BIGINT</td>
<td>evmon_waits_total - Event monitor total waits monitor element</td>
</tr>
<tr>
<td>TOTAL_EXTENDED_LATCH_WAIT_TIME</td>
<td>BIGINT</td>
<td>total_extended_latch_wait_time - Total extended latch wait time monitor element</td>
</tr>
<tr>
<td>TOTAL_EXTENDED_LATCH_WAITS</td>
<td>BIGINT</td>
<td>total_extended_latch_waits - Total extended latch waits monitor element</td>
</tr>
<tr>
<td>TOTAL_DISP_RUN_QUEUE_TIME</td>
<td>BIGINT</td>
<td>total_disp_run_queue_time - Total dispatcher run queue time monitor element</td>
</tr>
<tr>
<td>TOTAL_PEDS</td>
<td>BIGINT</td>
<td>total_peds - Total partial early distincts monitor element</td>
</tr>
<tr>
<td>DISABLED_PEDS</td>
<td>BIGINT</td>
<td>disabled_peds - Disabled partial early distincts monitor element</td>
</tr>
<tr>
<td>POST_THRESHOLD_PEDS</td>
<td>BIGINT</td>
<td>post_threshold_peds - Partial early distincts threshold monitor element</td>
</tr>
<tr>
<td>TOTAL_PEAS</td>
<td>BIGINT</td>
<td>total_peas - Total partial early aggregations monitor element</td>
</tr>
<tr>
<td>POST_THRESHOLD_PEAS</td>
<td>BIGINT</td>
<td>post_threshold_peas - Partial early aggregation threshold monitor element</td>
</tr>
<tr>
<td>TQ_SORT_HEAP_REQUESTS</td>
<td>BIGINT</td>
<td>tq_sort_heap_requests - Table queue sort heap requests monitor element</td>
</tr>
<tr>
<td>TQ_SORT_HEAP_REJECTIONS</td>
<td>BIGINT</td>
<td>tq_sort_heap_rejections - Table queue sort heap rejections monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_DATA_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_data_reqs - Data prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_INDEX_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_index_reqs - Index prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_XDA_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_xda_reqs - XDA prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_DATA_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_data_reqs - Data prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_INDEX_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_index_reqs - Index prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_XDA_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_xda_reqs - XDA data prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_OTHER_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_other_reqs - Other requests handled by prefetchers monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_DATA_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_data_pages - Data pages prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_INDEX_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_index_pages - Index pages prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_XDA_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_xda_pages - XDA pages prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_DATA_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_data_pages - Data pages prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_INDEX_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_index_pages - Index pages prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_XDA_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_xda_pages - XDA data pages prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_DATA_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_data_reqs - Failed data prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_INDEX_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_index_reqs - Failed index prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_XDA_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_xda_reqs - Failed XDA prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_TEMP_DATA_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_temp_data_reqs - Failed data prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_TEMP_INDEX_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_temp_index_reqs - Failed index prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_TEMP_XDA_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_temp_xda_reqs - Failed XDA prefetch requests for temporary table spaces monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_OTHER_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_other_reqs - Failed non-prefetch requests monitor element</td>
</tr>
<tr>
<td>PREFETCH_WAIT_TIME</td>
<td>BIGINT</td>
<td>prefetch_wait_time - Time waited for prefetch monitor element</td>
</tr>
<tr>
<td>PREFETCH_WAITS</td>
<td>BIGINT</td>
<td>prefetch_waits - Prefetcher wait count monitor element</td>
</tr>
<tr>
<td>POOL_DATA_GBP_INDEP_PAGES_FOUND_IN_LBP</td>
<td>BIGINT</td>
<td>pool_data_gbp_indep_pages_found_in_lbp - Group buffer pool independent data pages found in local buffer pool monitor element</td>
</tr>
<tr>
<td>POOL_INDEX_GBP_INDEP_PAGES_FOUND_IN_LBP</td>
<td>BIGINT</td>
<td>pool_index_gbp_indep_pages_found_in_lbp - Group buffer pool independent index pages found in local buffer pool monitor element</td>
</tr>
<tr>
<td>POOL_XDA_GBP_INDEP_PAGES_FOUND_IN_LBP</td>
<td>BIGINT</td>
<td>pool_xda_gbp_indep_pages_found_in_lbp - Group buffer pool XDA independent pages found in local buffer pool monitor element</td>
</tr>
<tr>
<td>IDA_SEND_WAIT_TIME</td>
<td>BIGINT</td>
<td>ida_send_wait_time - Time spent waiting to send data monitor element</td>
</tr>
<tr>
<td>IDA SENDS_TOTAL</td>
<td>BIGINT</td>
<td>ida_sends_total - Number of times data sent monitor element</td>
</tr>
<tr>
<td>IDA_SEND_VOLUME</td>
<td>BIGINT</td>
<td>ida_send_volume - Total data volume sent monitor element</td>
</tr>
<tr>
<td>IDA_RECV_WAIT_TIME</td>
<td>BIGINT</td>
<td>ida_recv_wait_time - Time spent waiting to receive data monitor element</td>
</tr>
<tr>
<td>IDA_RECVS_TOTAL</td>
<td>BIGINT</td>
<td>ida_recvs_total - Number of times data received monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IDA_RECV_VOLUME</td>
<td>BIGINT</td>
<td>ida_recv_volume - Total data volume received monitor element</td>
</tr>
<tr>
<td>ROWS_DELETED</td>
<td>BIGINT</td>
<td>rows_deleted - Rows deleted monitor element</td>
</tr>
<tr>
<td>ROWS_INSERTED</td>
<td>BIGINT</td>
<td>rows_inserted - Rows inserted monitor element</td>
</tr>
<tr>
<td>ROWS_UPDATED</td>
<td>BIGINT</td>
<td>rows_updated - Rows updated monitor element</td>
</tr>
<tr>
<td>TOTAL_HASH_JOINS</td>
<td>BIGINT</td>
<td>total_hash_joins - Total Hash Joins monitor element</td>
</tr>
<tr>
<td>TOTAL_HASH_LOOPS</td>
<td>BIGINT</td>
<td>total_hash_loops - Total Hash Loops monitor element</td>
</tr>
<tr>
<td>HASH_JOIN_OVERFLOWS</td>
<td>BIGINT</td>
<td>hash_join_overflows - Hash Join Overflows monitor element</td>
</tr>
<tr>
<td>HASH_JOIN_SMALL_OVERFLOWS</td>
<td>BIGINT</td>
<td>hash_join_small_overflows - Hash Join Small Overflows monitor element</td>
</tr>
<tr>
<td>POST_SHRTHRESHOLD_HASH_JOINS</td>
<td>BIGINT</td>
<td>post_shrthreshold_hash_joins - Post threshold hash joins monitor element</td>
</tr>
<tr>
<td>TOTAL_OLAP_FUNCS</td>
<td>BIGINT</td>
<td>total_olap_funcs - Total OLAP Functions monitor element</td>
</tr>
<tr>
<td>OLAP_FUNC_OVERFLOWS</td>
<td>BIGINT</td>
<td>olap_func_overflows - OLAP Function Overflows monitor element</td>
</tr>
<tr>
<td>INT_ROWS_DELETED</td>
<td>BIGINT</td>
<td>int_rows_deleted - Internal Rows Deleted monitor element</td>
</tr>
<tr>
<td>INT_ROWS_INSERTED</td>
<td>BIGINT</td>
<td>int_rows_inserted - Internal Rows Inserted monitor element</td>
</tr>
<tr>
<td>INT_ROWS_UPDATED</td>
<td>BIGINT</td>
<td>int_rows_updated - Internal Rows Updated monitor element</td>
</tr>
<tr>
<td>POOL_COL_L_READS</td>
<td>BIGINT</td>
<td>pool_col_l_reads - Buffer pool column-organized logical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_COL_L_READS</td>
<td>BIGINT</td>
<td>pool_temp_col_l_reads - Buffer pool column-organized temporary logical reads monitor element</td>
</tr>
<tr>
<td>POOL_COL_P_READS</td>
<td>BIGINT</td>
<td>pool_col_p_reads - Buffer pool column-organized physical reads monitor element</td>
</tr>
<tr>
<td>POOL_TEMP_COL_P_READS</td>
<td>BIGINT</td>
<td>pool_temp_col_p_reads - Buffer pool column-organized temporary physical reads monitor element</td>
</tr>
<tr>
<td>POOL_COL_LBP_PAGES_FOUND</td>
<td>BIGINT</td>
<td>pool_col_lbp_pages_found - Buffer pool column-organized LBP pages found monitor element</td>
</tr>
<tr>
<td>POOL_COL_WRITES</td>
<td>BIGINT</td>
<td>pool_col_writes - Buffer pool column-organized writes monitor element</td>
</tr>
<tr>
<td>POOL_COL_GBP_L_READS</td>
<td>BIGINT</td>
<td>pool_col_gbp_l_reads - Buffer pool column-organized GBP logical reads monitor element</td>
</tr>
<tr>
<td>POOL_COL_GBP_P_READS</td>
<td>BIGINT</td>
<td>pool_col_gbp_p_reads - Buffer pool column-organized GBP physical reads monitor element</td>
</tr>
<tr>
<td>POOL_COL_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>pool_col_gbp_invalid_pages - Buffer pool column-organized GBP invalid data pages monitor element</td>
</tr>
<tr>
<td>POOL_COL_GBP_INDEP_PAGES_FOUND_IN_LBP</td>
<td>BIGINT</td>
<td>pool_col_gbp_indep_pages_found_in_lbp - Buffer pool column-organized GBP independent pages found in local buffer pool monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_COLREQS</td>
<td>BIGINT</td>
<td>pool_queued_async_colreqs - Column-organized prefetch requests monitor element</td>
</tr>
</tbody>
</table>
Table 85. Information returned by the MON_GET_ACTIVITY routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_COL_REQS</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_col_reqs - Column-organized temporary prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_COL_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_col_pages - Column-organized page prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_QUEUED_ASYNC_TEMP_COL_PAGES</td>
<td>BIGINT</td>
<td>pool_queued_async_temp_col_pages - Column-organized page temporary prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_COL_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_col_reqs - Failed column-organized prefetch requests monitor element</td>
</tr>
<tr>
<td>POOL_FAILED_ASYNC_TEMP_COL_REQS</td>
<td>BIGINT</td>
<td>pool_failed_async_temp_col_reqs - Failed column-organized prefetch requests monitor element</td>
</tr>
<tr>
<td>TOTAL_COL_TIME</td>
<td>BIGINT</td>
<td>total_col_time - Total column-organized time monitor element</td>
</tr>
<tr>
<td>TOTAL_COL_PROC_TIME</td>
<td>BIGINT</td>
<td>total_col_proc_time - Total column-organized processing time monitor element</td>
</tr>
<tr>
<td>TOTAL_COL_EXECUTIONS</td>
<td>BIGINT</td>
<td>total_col_executions - Total column-organized executions monitor element</td>
</tr>
<tr>
<td>COMM_EXIT_WAIT_TIME</td>
<td>BIGINT</td>
<td>comm_exit_wait_time - Communication buffer exit wait time</td>
</tr>
<tr>
<td>COMM_EXIT_WAITS</td>
<td>BIGINT</td>
<td>comm_exit_waits - Communication buffer exit number of waits</td>
</tr>
<tr>
<td>POST_THRESHOLD_HASH_JOINS</td>
<td>BIGINT</td>
<td>post_threshold_hash_joins - Hash Join Threshold monitor element</td>
</tr>
<tr>
<td>POOL_DATA_CACHING_TIER_L_READS</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_INDEX_CACHING_TIER_L_READS</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_XDA_CACHING_TIER_L_READS</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_COL_CACHING_TIER_L_READS</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_DATA_CACHING_TIER_PAGE_WRITES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_INDEX_CACHING_TIER_PAGE_WRITES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_XDA_CACHING_TIER_PAGE_WRITES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_COL_CACHING_TIER_PAGE_UPDATES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_DATA_CACHING_TIER_PAGE_UPDATES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_INDEX_CACHING_TIER_PAGE_UPDATES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_XDA_CACHING_TIER_PAGE_UPDATES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_COL_CACHING_TIER_PAGE_UPDATES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_CACHING_TIER_PAGE_READ_TIME</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>
Table 85. Information returned by the MON_GET_ACTIVITY routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POOL_CACHING_TIER_PAGE_WRITE_TIME</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_DATA_CACHING_TIER_PAGES_FOUND</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_INDEX_CACHING_TIER_PAGES_FOUND</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_XDA_CACHING_TIER_PAGES_FOUND</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_COL_CACHING_TIER_PAGES_FOUND</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_DATA_CACHING_TIER_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_INDEX_CACHING_TIER_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_XDA_CACHING_TIER_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>POOL_COL_CACHING_TIER_GBP_INVALID_PAGES</td>
<td>BIGINT</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>TOTAL_HASH_GRPBY</td>
<td>BIGINT</td>
<td>total_hash_grpbys - Total hash group by operations</td>
</tr>
<tr>
<td>HASH_GRPBY_OVERFLOW</td>
<td>BIGINT</td>
<td>hash_grpby_overflows - Hash group by overflows</td>
</tr>
<tr>
<td>POST_THRESHOLD_HASH_GRPBY</td>
<td>BIGINT</td>
<td>post_threshold_hash_grpbys - Hash group by threshold</td>
</tr>
<tr>
<td>POST_THRESHOLD_OPAFUNKS</td>
<td>BIGINT</td>
<td>post_threshold_oapfuncs - OLAP function threshold monitor element</td>
</tr>
<tr>
<td>SEMANTIC_ENV_ID</td>
<td>BIGINT</td>
<td>semantic_env_id - Query semantic compilation environment ID monitor element</td>
</tr>
<tr>
<td>STMTID</td>
<td>BIGINT</td>
<td>stmtid - Query statement ID monitor element</td>
</tr>
<tr>
<td>PLANID</td>
<td>BIGINT</td>
<td>planid - Query plan ID monitor element</td>
</tr>
<tr>
<td>POST_THRESHOLD_COL_VECTOR_CONSUMERS</td>
<td>BIGINT</td>
<td>post_threshold_col_vector_consumers - Post threshold columnar_vector consumers</td>
</tr>
<tr>
<td>TOTAL_COL_VECTOR_CONSUMERS</td>
<td>BIGINT</td>
<td>total_col_vector_consumers - Total columnar_vector consumers</td>
</tr>
<tr>
<td>ACTIVE_HASH_GRPBY</td>
<td>BIGINT</td>
<td>active_hash_grpbys - Active hash GROUP BY operations monitor element</td>
</tr>
<tr>
<td>ACTIVE_HASH_GRPBY_TOP</td>
<td>BIGINT</td>
<td>active_hash_grpbys_top - Active hash GROUP BY operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_HASH_JOINS</td>
<td>BIGINT</td>
<td>active_hash_joins - Active hash joins monitor element</td>
</tr>
<tr>
<td>ACTIVE_HASH_JOINS_TOP</td>
<td>BIGINT</td>
<td>active_hash_joins_top - Active hash joins operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_OPAFUNKS</td>
<td>BIGINT</td>
<td>active_oapfuncs - Active OLAP Functions monitor element</td>
</tr>
<tr>
<td>ACTIVE_OPAFUNKS_TOP</td>
<td>BIGINT</td>
<td>active_oapfuncs_top - Active OLAP functions operations high watermark</td>
</tr>
</tbody>
</table>
### Table 85. Information returned by the MON_GET_ACTIVITY routine (continued)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE_PEAS</td>
<td>BIGINT</td>
<td>active_peas - Active partial early aggregations operations</td>
</tr>
<tr>
<td>ACTIVE_PEAS_TOP</td>
<td>BIGINT</td>
<td>active_peas_top - Active partial early aggregations operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_PEDS</td>
<td>BIGINT</td>
<td>active_peds - Active partial early distinct operations</td>
</tr>
<tr>
<td>ACTIVE_PEDS_TOP</td>
<td>BIGINT</td>
<td>active_peds_top - Active partial early distinct operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_SORT_CONSUMERS</td>
<td>BIGINT</td>
<td>active_sort_consumers - Active sort consumers</td>
</tr>
<tr>
<td>ACTIVE_SORT_CONSUMERS_TOP</td>
<td>BIGINT</td>
<td>active_sort_consumers_top - Active sort consumers high watermark</td>
</tr>
<tr>
<td>ACTIVE_SORTS</td>
<td>BIGINT</td>
<td>active_sorts - Active Sorts monitor element</td>
</tr>
<tr>
<td>ACTIVE_SORTS_TOP</td>
<td>BIGINT</td>
<td>active_sorts_top - Active Sorts high watermark</td>
</tr>
<tr>
<td>ACTIVE_COL_VECTOR_CONSUMERS</td>
<td>BIGINT</td>
<td>active_col_vector_consumers - Active columnar_vector consumers</td>
</tr>
<tr>
<td>ACTIVE_COL_VECTOR_CONSUMERS_TOP</td>
<td>BIGINT</td>
<td>active_col_vector_consumers_top - Active columnar_vector consumers high watermark</td>
</tr>
<tr>
<td>SORT_CONSUMER_HEAP_TOP</td>
<td>BIGINT</td>
<td>sort_consumer_heap_top - Individual private sort heap consumer high watermark</td>
</tr>
<tr>
<td>SORT_CONSUMER_SHRHEAP_TOP</td>
<td>BIGINT</td>
<td>sort_consumer_shrheap_top - Individual shared sort heap consumer high watermark</td>
</tr>
<tr>
<td>SORT_HEAP_ALLOCATED</td>
<td>BIGINT</td>
<td>sort_heap_allocated - Total Sort Heap Allocated monitor element</td>
</tr>
<tr>
<td>SORT_HEAP_Top</td>
<td>BIGINT</td>
<td>sort_heap_top - Sort private heap high watermark monitor element</td>
</tr>
<tr>
<td>SORT_SHRHEAP_ALLOCATED</td>
<td>BIGINT</td>
<td>sort_shrheap_allocated - Sort Share Heap Currently Allocated monitor element</td>
</tr>
<tr>
<td>SORT_SHRHEAP_TOP</td>
<td>BIGINT</td>
<td>sort_shrheap_top - Sort share heap high watermark monitor element</td>
</tr>
<tr>
<td>TOTAL_INDEX_BUILD_TIME</td>
<td>BIGINT</td>
<td>total_index_build_time - Total time spent building indexes due to index creation</td>
</tr>
<tr>
<td>TOTAL_INDEX_BUILD_PROC_TIME</td>
<td>BIGINT</td>
<td>total_indexbuild_proc_time - Total non-wait time spent building indexes due to index creation</td>
</tr>
<tr>
<td>TOTAL_INDEXES_BUILT</td>
<td>BIGINT</td>
<td>total_indexes_built - Total number of indexes built</td>
</tr>
</tbody>
</table>

### Examples

**Activities currently running with a known application handle**

After you identify the application handle, you can look up all the activities currently running in this application. For example, suppose that an administrator wants to list the activities of an application whose application handle, determined by using the `LIST APPLICATIONS` command, is 1. The administrator runs the following query:

```sql
SELECT SUBSTR(CHAR(COORD_MEMBER),1,5) AS COORD,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       SUBSTR(CHAR(UOW_ID),1,5) AS UOWID,
       SUBSTR(CHAR(ACTIVITY_ID),1,5) AS ACTID,
       SUBSTR(CHAR(PARENT_UOW_ID),1,8) AS PARUOWID,
       SUBSTR(CHAR(PARENT_ACTIVITY_ID),1,8) AS PARACTID,
       ACTIVITY_TYPE AS ACTTYPE,
       SUBSTR(CHAR(NESTING_LEVEL),1,7) AS NESTING
FROM TABLE(MON_GET_ACTIVITY(1, -2)) AS WLOACTS
ORDER BY MEMB, UOWID, ACTID
```
Sample output from the query is as follows:

<table>
<thead>
<tr>
<th>COORD</th>
<th>MEMB</th>
<th>UOWID</th>
<th>ACTID</th>
<th>PARUOWID</th>
<th>PARACTID</th>
<th>ACTTYPE</th>
<th>NESTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>CALL</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>READ_DML</td>
<td>1</td>
</tr>
</tbody>
</table>

Activities currently running on the system

The following query uses MON_GET_ACTIVITY to provide statement text for all SQL activities currently running on the system:

```sql
SELECT t.application_handle, t.uow_id, t.activity_id,
       VARCHAR(t.stmt_text, 256) as stmt_text
FROM table(MON_GET_ACTIVITY(NULL, -1)) as t
```

Sample output is as follows:

<table>
<thead>
<tr>
<th>APPLICATION_HANDLE</th>
<th>UOW_ID</th>
<th>ACTIVITY_ID</th>
<th>STMT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>SELECT * FROM SYSCAT.TABLES</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td>36</td>
<td>INSERT INTO T1 VALUES(123)</td>
</tr>
</tbody>
</table>

**MON_GET_WORKLOAD_STATS table function - Return workload statistics**

The MON_GET_WORKLOAD_STATS function returns one row of workload statistics for every combination of workload name and database member number.

**Authorization**

One of the following authorizations is required:

- EXECUTE privilege on the routine
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- WLMADM authority

**Default PUBLIC privilege**

None

**Syntax**

```
MON_GET_WORKLOAD_STATS(workload_name, member)
```

The schema is SYSPROC.

**Routine parameters**

*workload_name*

An input argument of type VARCHAR(128) that specifies a workload for which the statistics are to be returned. If the argument is NULL or an empty string, statistics are returned for all workloads.

*member*

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

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKLOAD_NAME</td>
<td>VARCHAR(128)</td>
<td>workload_name - Workload name monitor element</td>
</tr>
<tr>
<td>DBPARTITIONNUM</td>
<td>SMALLINT</td>
<td>dbpartitionnum - Database partition number monitor element</td>
</tr>
<tr>
<td>LAST_RESET</td>
<td>TIMESTAMP</td>
<td>last_reset - Last Reset Timestamp monitor element</td>
</tr>
<tr>
<td>CONCURRENT_WLO_TOP</td>
<td>INTEGER</td>
<td>concurrent_wlo_top - Concurrent workload occurrences top monitor element</td>
</tr>
<tr>
<td>CONCURRENT_WLO_ACT_TOP</td>
<td>INTEGER</td>
<td>concurrent_wlo_act_top - Concurrent WLO activity top monitor element</td>
</tr>
<tr>
<td>COORD_ACT_COMPLETED_TOTAL</td>
<td>BIGINT</td>
<td>coord_act_completed_total - Coordinator activities completed total monitor element</td>
</tr>
<tr>
<td>COORD_ACT_ABORTED_TOTAL</td>
<td>BIGINT</td>
<td>coord_act_aborted_total - Coordinator activities aborted total monitor element</td>
</tr>
<tr>
<td>COORD_ACT_REJECTED_TOTAL</td>
<td>BIGINT</td>
<td>coord_act_rejected_total - Coordinator activities rejected total monitor element</td>
</tr>
<tr>
<td>WLO_COMPLETED_TOTAL</td>
<td>BIGINT</td>
<td>wlo_completed_total - Workload occurrences completed total monitor element</td>
</tr>
<tr>
<td>COORD_ACT_LIFETIME_TOP</td>
<td>BIGINT</td>
<td>coord_act_lifetime_top - Coordinator activity lifetime top monitor element</td>
</tr>
<tr>
<td>COORD_ACT_LIFETIME_AVG</td>
<td>DOUBLE</td>
<td>coord_act_lifetime_avg - Coordinator activity lifetime average monitor element</td>
</tr>
<tr>
<td>COORD_ACT_LIFETIME_STDDEV</td>
<td>DOUBLE</td>
<td>Standard deviation of lifetime for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null. This standard deviation is computed from the coordinator activity lifetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. If any values fall into the last histogram bin, the value -1 is returned.</td>
</tr>
<tr>
<td>COORD_ACT_EXEC_TIME_AVG</td>
<td>DOUBLE</td>
<td>coord_act_exec_time_avg - Coordinator activities execution time average monitor element</td>
</tr>
<tr>
<td>COORD_ACT_EXEC_TIME_STDDEV</td>
<td>DOUBLE</td>
<td>Standard deviation of the execution times for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. This standard deviation is computed from the coordinator activity executetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. If any values fall into the last histogram bin, the value -1 is returned. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null.</td>
</tr>
<tr>
<td>COORD_ACT_QUEUE_TIME_AVG</td>
<td>DOUBLE</td>
<td>coord_act_queue_time_avg - Coordinator activity queue time average monitor element</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COORD_ACT_QUEUE_TIME_STDDEV</td>
<td>DOUBLE</td>
<td>Standard deviation of the queue time for completed or aborted coordinator activities at nesting level 0 that are associated with this workload. Units are milliseconds. If the COLLECT AGGREGATE ACTIVITY DATA parameter of the workload is set to NONE, the value of the column is null. This standard deviation is computed from the coordinator activity queuetime histogram and may be inaccurate if the histogram is not correctly sized to fit the data. If any values fall into the last histogram bin, the value -1 is returned.</td>
</tr>
<tr>
<td>UOW_TOTAL_TIME_TOP</td>
<td>BIGINT</td>
<td>uow_total_time_top - UOW total time top monitor element</td>
</tr>
<tr>
<td>UOW_THROUGHPUT</td>
<td>DOUBLE</td>
<td>uow_throughput - Unit of work throughput monitor element</td>
</tr>
<tr>
<td>UOW_LIFETIME_AVG</td>
<td>DOUBLE</td>
<td>uow_lifetime_avg - Unit of work lifetime average monitor element</td>
</tr>
<tr>
<td>UOW_COMPLETED_TOTAL</td>
<td>BIGINT</td>
<td>uow_completed_total - Total completed units of work monitor element</td>
</tr>
<tr>
<td>TOTAL_CPU_TIME</td>
<td>BIGINT</td>
<td>total_cpu_time - Total CPU time monitor element</td>
</tr>
<tr>
<td>TOTAL_DISP_RUN_QUEUE_TIME</td>
<td>BIGINT</td>
<td>total_disp_run_queue_time - Total dispatcher run queue time monitor element</td>
</tr>
<tr>
<td>ACT_THROUGHPUT</td>
<td>DOUBLE</td>
<td>act_throughput - Activity throughput monitor element</td>
</tr>
<tr>
<td>CPU_UTILIZATION</td>
<td>DOUBLE</td>
<td>cpu_utilization - CPU utilization monitor element</td>
</tr>
<tr>
<td>APP_ACT_COMPLETED_TOTAL</td>
<td>BIGINT</td>
<td>app_act_completed_total - Total successful external coordinator activities monitor element</td>
</tr>
<tr>
<td>APP_ACT_ABORTED_TOTAL</td>
<td>BIGINT</td>
<td>app_act_aborted_total - Total failed external coordinator activities monitor element</td>
</tr>
<tr>
<td>APP_ACT_REJECTED_TOTAL</td>
<td>BIGINT</td>
<td>app_act_rejected_total - Total rejected external coordinator activities monitor element</td>
</tr>
<tr>
<td>MEMBER</td>
<td>SMALLINT</td>
<td>member - Database member monitor element</td>
</tr>
<tr>
<td>ACTIVE_HASH_GRPBYS_TOP</td>
<td>BIGINT</td>
<td>active_hash_grpby_top - Active hash Group By operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_HASH_JOINS_TOP</td>
<td>BIGINT</td>
<td>active_hash_joins_top - Active hash joins operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_OOLAP_FUNCS_TOP</td>
<td>BIGINT</td>
<td>active_oolap_fun_top - Active OLAP functions operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_PEAS_TOP</td>
<td>BIGINT</td>
<td>active_peas_top - Active partial early aggregations operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_PEDS_TOP</td>
<td>BIGINT</td>
<td>active_peds_top - Active partial early distinct operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_SORT_CONSUMERS_TOP</td>
<td>BIGINT</td>
<td>active_sort_consumers_top - Active sort consumers operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_SORTS_TOP</td>
<td>BIGINT</td>
<td>active_sorts_top - Active Sorts operations high watermark</td>
</tr>
<tr>
<td>ACTIVE_COL_VECTOR_CONSUMERS_TOP</td>
<td>BIGINT</td>
<td>active_col_vector_consumers_top - Active columnar vector consumers operations high watermark</td>
</tr>
</tbody>
</table>
### Usage notes

The function does not aggregate data across workloads, members, or service classes. However, you can use SQL queries to aggregate data.

### Example

The following query displays statistics for workloads:

```sql
SELECT SUBSTR(WORKLOAD_NAME,1,18) AS WL_DEF_NAME,
       SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
       COORD_ACT_LIFETIME_TOP,
       COORD_ACT_LIFETIME_AVG,
       COORD_ACT_LIFETIME_STDDEV
FROM TABLE(MON_GET_WORKLOAD_STATS(CAST(NULL AS VARCHAR(128)), -2)) AS WLSTATS
ORDER BY WL_DEF_NAME, MEMB
```

Sample output from the query is as follows:

```
<table>
<thead>
<tr>
<th>WL_DEF_NAME</th>
<th>MEMB</th>
<th>COORD_ACT_LIFETIME_TOP</th>
<th>COORD_ACT_LIFETIME_AVG</th>
<th>COORD_ACT_LIFETIME_STDDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSDEFAULTADMWKLT</td>
<td>0</td>
<td>-1</td>
<td>-1.00000000000000E+00</td>
<td>-1.00000000000000E+00</td>
</tr>
<tr>
<td>SYSDEFAULTUSERWK</td>
<td>0</td>
<td>-1</td>
<td>-1.00000000000000E+00</td>
<td>-1.00000000000000E+00</td>
</tr>
<tr>
<td>WL1</td>
<td>0</td>
<td>2</td>
<td>2.56000000000000E+00</td>
<td>6.00000000000000E+01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### WLM_SET_CLIENT_INFO procedure - Set client information

The `WLM_SET_CLIENT_INFO` procedure sets client information associated with the current connection at the DB2 server.

By using this procedure, you can set the client's user ID, application name, workstation name, accounting information, or workload information at the DB2 server. Calling this procedure changes the stored values of the relevant transaction processor (TP) monitor client information fields and special register settings for this connection.

The client information fields are used at the DB2 server for determining the identity of the application or user currently using the connection. The client information fields for a connection are considered during DB2 workload evaluation and also displayed in any DB2 audit records or application snapshots generated for this connection.

Unlike the sqleseti API, this procedure does not set client information at the client but instead sets the corresponding client attributes on the DB2 server. Therefore,
you cannot use the sqleqry API to query the client information that is set at the DB2 server using this procedure. If an application uses the sqleseti API to change the client information, the new values will change the setting at the DB2 server. If the sqleseti API is used to change either the user ID or the application name without changing the accounting information, the accounting information at the DB2 server will also be reset to the value of the accounting information at the client.

The data values provided with the procedure are converted to the appropriate database code page before being stored in the related TP monitor fields or special registers. Any data value which exceeds the maximum supported size after conversion to the database code page is truncated before being stored at the server. The truncated values are returned by both the TP monitor fields and the special registers when those stored values are queried.

The WLM_SET_CLIENT_INFO procedure is not under transaction control, and client information changes made by the procedure are independent of committing or rolling back units of work. However, because workload reevaluation occurs at the beginning of the next unit of work for each application, you must issue either a COMMIT or a ROLLBACK statement to make client information changes effective.

Syntax

```
WLM_SET_CLIENT_INFO(client_userid, client_wrkstnname, client_applname, client_acctstr, client_workload)
```

The schema is SYSPROC.

Procedure parameters

**client_userid**

An input argument of type VARCHAR(255) that specifies the user ID for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the user ID for the client is reset to the default value, which is blank.

**client_wrkstnname**

An input argument of type VARCHAR(255) that specifies the workstation name for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the workstation name for the client is reset to the default value, which is blank.

**client_applname**

An input argument of type VARCHAR(255) that specifies the application name for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the application name for the client is reset to the default value, which is blank.

**client_acctstr**

An input argument of type VARCHAR(255) that specifies the accounting string for the client. If you specify NULL, the value remains unchanged. If you specify an empty string, which is the default value, the accounting string for the client is reset to the default value, which is blank.
An input argument of type VARCHAR(255) that specifies the workload assignment mode for the client. If you specify NULL, the value remains unchanged. The values are as follows:

SYSDEFAULTADMWORKLOAD
Specifies that the database connection will be assigned to SYSDEFAULTADMWORKLOAD, enabling users with ACCESSCTRL, DATAACCESS, DBADM, SECADM, or WLMADM authority to bypass the normal workload evaluation.

AUTOMATIC
Specifies that the database connection will be assigned to a workload chosen by the workload evaluation that is performed automatically by the server.

Note: The client_workload argument is case sensitive.

Authorization

One of the following authorities is required to execute the routine:
• EXECUTE privilege on the routine
• DATAACCESS authority
• DBADM authority
• SQLADM authority
• WLMADM authority

Default PUBLIC privilege
None

Examples

The following procedure call sets the user ID, workstation name, application name, accounting string, and workload assignment mode for the client:
CALL SYSPROC.WLM_SET_CLIENT_INFO('db2user', 'machine.torolab.ibm.com', 'auditor', 'Accounting department', 'AUTOMATIC')

The following procedure call sets the user ID to db2user2 for the client without setting the other client attributes:
CALL SYSPROC.WLM_SET_CLIENT_INFO('db2user2', NULL, NULL, NULL, NULL)

The following procedure call resets the user ID for the client to blank without modifying the values of the other client attributes:
CALL SYSPROC.WLM_SET_CLIENT_INFO('', NULL, NULL, NULL, NULL)

Usage notes

If the input that you specify for any procedure parameter exceeds its specified field length, the input field is truncated and the procedure runs using the truncated inputs.

Input fields containing single quotations are not supported and result in an error.
Workload management monitor elements

The following monitor elements provide information about activities, threshold violations, and workload management statistics.

**act_cpu_time_top - Activity CPU time top monitor element**

The high watermark for processor time used by activities at all nesting levels in a service class, workload, or work class. This value is reported in microseconds.

The monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class or workload in which the activity runs is set to NONE. Activities contribute toward this high watermark only when request metrics are enabled. If the collection of activity metrics is not enabled, a value of 0 is returned.

For service classes, when you remap activities between service subclasses with a REMAP ACTIVITY action, only the act_cpu_time_top high watermark of the service subclass where an activity completes is updated, provided that a new high watermark is reached. The act_cpu_time_top high watermarks of other service subclasses an activity is mapped to but does not complete in are unaffected.

**Table 87. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine the highest amount of processor time used by an activity on a member for a service class, workload, or work class during the time interval collected.

**act_exec_time - Activity execution time monitor element**

The act_exec_time element stores the time spent executing at this member, in microseconds.

For cursors, the execution time is the combined time for the open, the fetches, and the close. The time when the cursor is idle is not counted toward execution time. For routines, execution time is the start to end of routine invocation. The lifetimes of any cursors left open by routine (to return a result set) after the routine finishes are not counted toward the routine execution time. For all other activities, execution time is the difference between start time and stop time. In all cases, execution time does not include time spent initializing or queued.

**Table 88. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

This element can be used alone to know the elapsed time spent executing the activity by DB2 on each member. This element can also be used together with
time_started and time_completed monitor elements on the coordinator member to compute the idle time for cursor activities. You can use the following formula:
Cursor idle time = (time_completed - time_started) - act_exec_time

**act_remapped_in - Activities remapped in monitor element**
Count of the number of activities to be remapped into this service subclass since the last reset.

*Table 89. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
</tbody>
</table>

*Table 90. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**
Use this count to determine whether the remapping of activities into the service subclass is occurring as desired.

**act_remapped_out - Activities remapped out monitor element**
Count of the number of activities to be remapped out of this service subclass since the last reset.

*Table 91. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
</tbody>
</table>

*Table 92. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**
Use this count to determine whether the remapping of activities out of the service subclass is occurring as desired.

**act_rows_read_top - Activity rows read top monitor element**
The high watermark for the number of rows read by activities at all nesting levels in a service class, workload, or work class.

The monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class or workload in which the activity runs is set to NONE. Activities contribute toward this high watermark only when request metrics are enabled. If the collection of activity metrics is not enabled, a value of 0 is returned.
For service classes, when you remap activities between service subclasses with a REMAP ACTIVITY action only the act_rows_read_top high watermark of the service subclass where an activity completes is updated, provided that a new high watermark is reached. The act_rows_read_top high watermarks of service subclasses an activity is mapped to but does not complete in are unaffected.

Table 93. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine the highest number of rows read by an activity on a member for a service class, workload, or work class during the time interval collected.

**act_throughput - Activity throughput monitor element**

The rate at which coordinator activities are completed at any nesting level. Measured in coordinator activities per second.

Table 94. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS - Get sample service class metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_SAMPLE_WORKLOAD_METRICS - Get sample workload metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 95. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
### Usage

When returned by the WLM_GET_SERVICE_SUBCLASS_STATS or the WLM_GET_WORKLOAD_STATS function, this monitor element represents the activity throughput since the last reset of the statistics.

When returned by the MON_SAMPLE_SERVICE_CLASS_METRICS or the MON_SAMPLE_WORKLOAD_METRICS function, this monitor element represents the activity throughput since the function was executed.

**act_total - Activities total monitor element**

Total number of activities at any nesting level that had work actions corresponding to the specified work class applied to them since the last reset.

*Table 96. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_WORK_ACTION_SET_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORK_ACTION_SET_STATS</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 97. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

### Usage

Every time an activity has one or more work actions associated with a work class applied to it, a counter for the work class is updated. This counter is exposed using the act_total monitor element. The counter can be used to judge the effectiveness of the work action set (for example, how many activities have actions applied). It can also be used to understand the different types of activities on the system.

**activate_timestamp - Activate timestamp monitor element**

The time when an event monitor was activated.

*Table 98. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>event_activity</td>
<td>-</td>
</tr>
<tr>
<td>Activity</td>
<td>event_activitystmt</td>
<td>-</td>
</tr>
<tr>
<td>Activity</td>
<td>event_activityvals</td>
<td>-</td>
</tr>
<tr>
<td>Threshold Violations</td>
<td>event_thresholdviolations</td>
<td>-</td>
</tr>
</tbody>
</table>

### Usage

Use this element to correlate information returned by the previously mentioned event types.
**activity_collected - Activity collected monitor element**

This element indicates whether or not activity event monitor records are to be collected for a violated threshold.

*Table 99. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine whether to expect an activity event for the activity that violated the threshold to be written to the activity event monitor.

When an activity finishes or aborts and the activity event monitor is active at the time, if the value of this monitor element is 'Y', the activity that violated this threshold will be collected. If the value of this monitor element is 'N', it will not be collected.

**activity_id - Activity ID monitor element**

Counter which uniquely identifies an activity for an application within a given unit of work.

*Table 100. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GETSERVICECLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GETWORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 101. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locking</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitystmt</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activityvals</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity.
To uniquely identify an activity outside its unit of work, use the combination of activity_id and uow_id plus one of the following monitor elements: appl_id or agent_id.

**activity_secondary_id - Activity secondary ID monitor element**

The value for this element is incremented each time an activity record is written for the same activity.

For example, if an activity record is written once as a result of having called the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure and a second time when the activity ends, the element would have a value of 0 for the first record and 1 for the second record.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>-</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitystmt</td>
<td>-</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activityvals</td>
<td>-</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>

**Usage**

Use this element with activity_id, uow_id, and appl_id monitor elements to uniquely identify activity records when information about the same activity has been written to the activities event monitor multiple times.

For example, information about an activity would be sent to the activities event monitor twice in the following case:

- the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure was used to capture information about the activity while it was running
- information about the activity was collected when the activity completed, because the COLLECT ACTIVITY DATA clause was specified on the service class with which the activity is associated

**activity_type - Activity type monitor element**

The type of the activity.

**Table 103. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Table 104. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

The possible values are:
- LOAD
- READ_DML
- WRITE_DML
- DDL
- CALL
- OTHER

The value OTHER is returned for SET statements that do not perform SQL (for example, SET special register, or SET EVENT MONITOR STATE) and the LOCK TABLE statement.

**agg_temp_tablespace_top - Aggregate temporary table space top monitor element**

The `agg_temp_tablespace_top` monitor element stores the high watermark, in KB, for the aggregate temporary table space usage of DML activities at all nesting levels in a service class.

The aggregate is computed by summing the temporary table space usage across all activities in the service subclass, and this high watermark represents the highest value reached by this aggregate since the last reset. The monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class is set to NONE. An AGGSQLTEMPSPACE threshold must be defined and enabled for at least one service subclass in the same superclass as the subclass to which this record belongs, otherwise a value of 0 is returned.

Table 105. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scsstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine the highest aggregate DML activity system temporary table space usage reached on a member for a service subclass in the time interval collected.

**arm_correlator - Application response measurement correlator monitor element**

Identifier of a transaction in the Application Response Measurement (ARM) standard.
Table 106. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

This element can be used to link an activity collected by the activities event monitor to the applications associated with the activity, if such applications also support the Application Response Measurement (ARM) standard.

**bin_id - Histogram bin identifier monitor element**

The identifier of a histogram bin. The bin_id is unique within a histogram.

Table 107. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to distinguish bins within the same histogram.

**bottom - Histogram bin bottom monitor element**

The exclusive bottom end of the range of a histogram bin. The value of this monitor element is also the top inclusive end of the range of the previous histogram bin, if there is one.

Table 108. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element with the corresponding top element to determine the range of a bin within a histogram.

**concurrent_act_top - Concurrent activity top monitor element**

The high watermark for the concurrent activities (at any nesting level) in a service subclass since the last reset.

**Note**: This element monitors the highest concurrent execution of all activities, including those activities that do not participate in the CONCURRENTDBCOORDACTIVITIES threshold. For example, although CALL statements do not count toward the concurrency that is enforced by the CONCURRENTDBCOORDACTIVITIES threshold, they are included in the concurrent activity high watermark measurement.
Table 109. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - list workload occurrences</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
</tbody>
</table>

Table 110. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

Use this element to know the highest concurrency of activities (including nested activities) reached on a member for a service subclass in the time interval collected.

concurrent_connection_top - Concurrent connection top monitor element

High watermark for concurrent coordinator connections in this service class since the last reset. This field has the same value in every subclass of the same superclass.

Table 111. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUPERCLASS_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return statistics of service superclasses</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUPERCLASS_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return statistics of service superclasses</td>
<td></td>
</tr>
</tbody>
</table>

Table 112. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

This element may be useful in determining where to place thresholds on connection concurrency by showing where the current high watermark is. It is also useful for verifying that such a threshold is configured correctly and doing its job.
**concurrent_wlo_act_top - Concurrent WLO activity top monitor element**

High watermark for concurrent activities (at any nesting level) of any occurrence of this workload since the last reset.

*Table 113. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 114. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to know the highest number of concurrent activities reached on a member for any occurrence of this workload in the time interval collected.

**concurrent_wlo_top - Concurrent workload occurrences top monitor element**

The high watermark for the concurrent occurrences of a workload since the last reset.

*Table 115. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 116. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to know the highest concurrency of workload occurrences reached on a member for a workload in the time interval collected.
**concurrentdbcoordactivities_db_threshold_id** - Concurrent database coordinator activities database threshold ID monitor element

The ID of the CONCURRENTDBCOORDACTIVITIES database threshold that was applied to the activity.

**Table 117. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to understand which CONCURRENTDBCOORDACTIVITIES database threshold, if any, was applied to the activity.

**concurrentdbcoordactivities_db_threshold_queued** - Concurrent database coordinator activities database threshold queued monitor element

This monitor element returns '1' (Yes) to indicate that the activity was queued by the CONCURRENTDBCOORDACTIVITIES database threshold. A value of '0' (No) indicates that the activity was not queued.

**Table 118. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to understand if the activity was queued by the CONCURRENTDBCOORDACTIVITIES database threshold applied to the activity.

**concurrentdbcoordactivities_db_threshold_value** - Concurrent database coordinator activities database threshold value monitor element

This monitor element returns the upper bound of the CONCURRENTDBCOORDACTIVITIES database threshold that was applied to the activity.
### Usage

Use this element to understand the value of the CONCURRENTDBCOORDACTIVITIES database threshold applied to the activity, if any.

**concurrentdbcoordactivities_db_threshold_violated** - Concurrent database coordinator activities database threshold violated monitor element

This monitor element returns ‘1’ (Yes) to indicate that the activity violated the CONCURRENTDBCOORDACTIVITIES database threshold. A value of ‘0’ (No) indicates that the activity has not yet violated the threshold.

### Usage

Use this element to determine if the activity violated the CONCURRENTDBCOORDACTIVITIES database threshold that was applied to the activity.

**concurrentdbcoordactivities_subclass_threshold_id** - Concurrent database coordinator activities service subclass threshold ID monitor element

This monitor element returns the ID of the CONCURRENTDBCOORDACTIVITIES service subclass threshold that was applied to the activity.
### Table 121. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITYDETAILSTABLE function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

#### Usage

Use this element to understand which CONCURRENTDBCOORDACTIVITIES service subclass threshold, if any, was applied to the activity.

**concurrentdbcoordactivities_subclass_threshold_queued** - *Concurrent database coordinator activities service subclass threshold queued monitor element*

This monitor element returns '1' (Yes) to indicate that the activity was queued by the CONCURRENTDBCOORDACTIVITIES service subclass threshold. A value of '0' (No) indicates that the activity was not queued.

### Table 122. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILSTABLE function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

#### Usage

Use this element to understand if the activity was queued by the CONCURRENTDBCOORDACTIVITIES service subclass threshold applied to the activity.

**concurrentdbcoordactivities_subclass_threshold_value** - *Concurrent database coordinator activities service subclass threshold value monitor element*

This monitor element returns the upper bound of the CONCURRENTDBCOORDACTIVITIES service subclass threshold that was applied to the activity.

### Table 123. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILSTABLE function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
### Table 123. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to understand the value of the CONCURRENTDBCOORDACTIVITIES service subclass threshold applied to the activity, if any.

**concurrentdbcoordactivities_subclass_threshold_violated** - Concurrent database coordinator activities service subclass threshold violated monitor element

This monitor element returns '1' (Yes) to indicate that the activity violated the CONCURRENTDBCOORDACTIVITIES service subclass threshold. A value of '0' (No) indicates that the activity has not yet violated the threshold.

### Table 124. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine if the activity violated the CONCURRENTDBCOORDACTIVITIES service subclass threshold that was applied to the activity.

**concurrentdbcoordactivities superclass_threshold_id** - Concurrent database coordinator activities service superclass threshold ID monitor element

The ID of the CONCURRENTDBCOORDACTIVITIES_SUPERCLASS threshold that was applied to the activity.

### Table 125. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
**Usage**

Use this element to understand which CONCURRENTDBCOORDACTIVITIES service superclass threshold, if any, was applied to the activity.

**concurrentdbcoordactivities_superclass_threshold_queued** - Concurrent database coordinator activities service superclass threshold queued monitor element

This monitor element returns '1' (Yes) to indicate that the activity was queued by the CONCURRENTDBCOORDACTIVITIES service superclass threshold. A value of '0' (No) indicates that the activity was not queued.

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 126. Table Function Monitoring Information**

**Usage**

Use this element to understand if the activity was queued by the CONCURRENTDBCOORDACTIVITIES service superclass threshold applied to the activity.

**concurrentdbcoordactivities_superclass_threshold_value** - Concurrent database coordinator activities service superclass threshold value monitor element

The upper bound of the CONCURRENTDBCOORDACTIVITIES service superclass threshold that was applied to the activity.

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 127. Table Function Monitoring Information**

**Usage**

Use this element to understand the value of the CONCURRENTDBCOORDACTIVITIES service superclass threshold applied to the activity, if any.
**Concurrent database coordinator activities service superclass threshold violated monitor element**

This monitor element returns '1' (Yes) to indicate that the activity violated the CONCURRENTDBCOORDACTIVITIES service superclass threshold. A value of '0' (No) indicates that the activity has not yet violated the threshold.

**Table 128. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine if the activity violated the CONCURRENTDBCOORDACTIVITIES service superclass threshold that was applied to the activity.

**Concurrent database coordinator activities workload work action set threshold ID monitor element**

The identifier of the CONCURRENTDBCOORDACTIVITIES workload work action set threshold that was applied to the activity.

**Table 129. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to understand which CONCURRENTDBCOORDACTIVITIES workload work action set threshold, if any, was applied to the activity.

**Concurrent database coordinator activities workload work action set threshold queued monitor element**

This monitor element returns '1' (Yes) to indicate that the activity was queued by the CONCURRENTDBCOORDACTIVITIES workload work action set threshold. A value of '0' (No) indicates that the activity was not queued.
Table 130. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of</td>
<td>Always collected</td>
</tr>
<tr>
<td>activities</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get</td>
<td>Always collected</td>
</tr>
<tr>
<td>complete activity details (reported in DETAILS XML</td>
<td></td>
</tr>
<tr>
<td>document)</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return a list of activities</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

Use this element to understand if the activity was queued by the CONCURRENTDBCOORDACTIVITIES workload work action set threshold applied to the activity.

**concurrentdbcoordactivities_wl_was_threshold_value**

Concurrent database coordinator activities workload work action set threshold value monitor element

The upper bound of the CONCURRENTDBCOORDACTIVITIES workload work action set threshold that was applied to the activity.

Table 131. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of</td>
<td>Always collected</td>
</tr>
<tr>
<td>activities</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get</td>
<td>Always collected</td>
</tr>
<tr>
<td>complete activity details (reported in DETAILS XML</td>
<td></td>
</tr>
<tr>
<td>document)</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return a list of activities</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

Use this element to understand the value of the CONCURRENTDBCOORDACTIVITIES workload work action set threshold applied to the activity.

**concurrentdbcoordactivities_wl_was_threshold_violated**

Concurrent database coordinator activities workload work action set threshold violated monitor element

This monitor element returns ‘1’ (Yes) to indicate that the activity violated the CONCURRENTDBCOORDACTIVITIES workload work action set threshold. A value of ‘0’ (No) indicates that the activity has not yet violated the threshold.

Table 132. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of</td>
<td>Always collected</td>
</tr>
<tr>
<td>activities</td>
<td></td>
</tr>
</tbody>
</table>
### Table 132. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

#### Usage

Use this element to determine if the activity violated the CONCURRENTDBCOORDACTIVITIES workload work action set threshold that was applied to the activity.

**coord_act_aborted_total - Coordinator activities aborted total monitor element**

The total number of coordinator activities at any nesting level that completed with errors since the last reset. For service classes, the value is updated when the activity completes. For workloads, the value is updated by each workload occurrence at the end of its unit of work.

For service classes, if you remap an activity to a different subclass with a REMAP ACTIVITY action before it aborts, then this activity counts only toward the total of the subclass it aborts in.

### Table 133. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

### Table 134. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_sclstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
**Usage**

Use this element to understand if activities on the system are completing successfully. Activities may be aborted due to cancellation, errors or reactive thresholds.

**coord_act_completed_total - Coordinator activities completed total monitor element**

The total number of coordinator activities at any nesting level that completed successfully since the last reset. For service classes, the value is updated when the activity completes. For workloads, the value is updated by each workload occurrence at the end of its unit of work.

For service classes, if you remap an activity to a different subclass with a REMAP ACTIVITY action before it completes, then this activity counts only toward the total of the subclass it completes in.

**Table 135. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 136. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

This element can be used to determine the throughput of activities in the system or to aid in calculating average activity lifetime across multiple members.

**coord_act_est_cost_avg - Coordinator activity estimated cost average monitor element**

Arithmetic mean of the estimated costs for coordinator DML activities at nesting level 0 associated with this service subclass or work class since the last reset.

If the internally tracked average has overflowed, the value -2 is returned. For service subclasses, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service subclass is set to NONE or BASE. For work
classes, this monitor element returns -1 if no COLLECT AGGREGATE ACTIVITY DATA EXTENDED work action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE or BASE. It is measured in units of timerons.

For service classes, the estimated cost of an activity is counted only toward the service subclass in which the activity enters the system. When you remap activities between service subclasses with a REMAP ACTIVITY action, the coord_act_est_cost_avg mean of the service subclass you remap an activity to is unaffected.

### Table 137. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scsstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

### Usage

Use this statistic to determine the arithmetic mean of the estimated costs of coordinator DML activities at nesting level 0 that are associated this service subclass, workload, or work class that completed or aborted since the last statistics reset.

This average can also be used to determine whether or not the histogram template used for the activity estimated cost histogram is appropriate. Compute the average activity estimated cost from the activity estimated cost histogram. Compare the computed average with this monitor element. If the computed average deviates from the true average reported by this monitor element, consider altering the histogram template for the activity estimated cost histogram, using a set of bin values that are more appropriate for your data.

**coord_act_exec_time_avg - Coordinator activities execution time average monitor element**

Arithmetic mean of execution times for coordinator activities at nesting level 0 associated with this service subclass or work class since the last reset.

If the internally tracked average has overflowed, the value -2 is returned. For service subclasses, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service subclass is set to NONE. For work classes, this monitor elements returns -1 if no COLLECT AGGREGATE ACTIVITY DATA work action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE. Units are milliseconds.

For service classes, when you remap activities between service subclasses with a REMAP ACTIVITY action, the coord_act_exec_time_avg mean of service subclasses an activity is mapped to but does not complete in is unaffected.
Table 139. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

Use this statistic to determine the arithmetic mean of execution time for coordinator activities associated with a service subclass, workload, or work class that completed or aborted.

This average can also be used to determine whether or not the histogram template used for the activity execution time histogram is appropriate. Compute the average activity execution time from the activity execution time histogram. Compare the computed average with this monitor element. If the computed average deviates from the true average reported by this monitor element, consider altering the histogram template for the activity execution time histogram, using a set of bin values that are more appropriate for your data.

coord_act_interarrival_time_avg - Coordinator activity arrival time average monitor element

Arithmetic mean of the time between arrivals of coordinator activities at nesting level 0 associated with this service subclass or work class since the last reset.

If the internally tracked average has overflowed, the value -2 is returned. For service subclasses, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service subclass is set to NONE or BASE. For work classes, this monitor elements returns -1 if no COLLECT AGGREGATE ACTIVITY DATA EXTENDED work action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE or BASE. It is measured in milliseconds.

For service classes, the inter-arrival time mean is calculated for service subclasses through which activities enter the system. When you remap activities between service subclasses with a REMAP ACTIVITY action, the coord_act_interarrival_time_avg of the service subclass you remap an activity to is unaffected.
Table 140. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_sctstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wctstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

Use this statistic to determine the arithmetic mean between arrivals of coordinator activities at nesting level 0 associated with this service subclass, workload, or work class.

The inter-arrival time can be used to determine arrival rate, which is the inverse of inter-arrival time. This average can also be used to determine whether or not the histogram template used for the activity inter-arrival time histogram is appropriate. Compute the average activity inter-arrival time from the activity inter-arrival time histogram. Compare the computed average with this monitor element. If the computed average deviates from the true average reported by this monitor element, consider altering the histogram template for the activity inter-arrival time histogram, using a set of bin values that are more appropriate for your data.

coord_act_lifetime_avg - Coordinator activity lifetime average monitor element

Arithmetic mean of lifetime for coordinator activities at nesting level 0 associated with this service subclass, workload, or work class since the last reset.

If the internally tracked average has overflowed, the value -2 is returned. For service subclasses, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service subclass is set to NONE. For work classes, this monitor element returns -1 if no COLLECT AGGREGATE ACTIVITY DATA work action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE. It is measured in milliseconds.

For service classes, when you remap activities between service subclasses with a REMAP ACTIVITY action, only the the coord_act_lifetime_avg mean of the final service class where the activity completes is affected.

Table 141. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>COLLECT AGGREGATE ACTIVITY DATA</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>COLLECT AGGREGATE ACTIVITY DATA</td>
</tr>
</tbody>
</table>
**Table 142. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this statistic to determine the arithmetic mean of the lifetime for coordinator activities associated with a service subclass, workload, or work class that completed or aborted.

This statistic can also be used to determine whether or not the histogram template used for the activity lifetime histogram is appropriate. Compute the average activity lifetime from the activity lifetime histogram. Compare the computed average with this monitor element. If the computed average deviates from the true average reported by this monitor element, consider altering the histogram template for the activity lifetime histogram, using a set of bin values that are more appropriate for your data.

**coord_act_lifetime_top - Coordinator activity lifetime top monitor element**

The coord_act_lifetime_top element is a high watermark for coordinator activity lifetime, counted at all nesting levels. The stored information is represented in milliseconds.

For service classes, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class is set to NONE. For work classes, this monitor element returns -1 if no COLLECT AGGREGATE ACTIVITY DATA work action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE.

To effectively use this statistic with service classes when you also remap activities between service subclasses with a REMAP ACTIVITY action, you must aggregate the coord_act_lifetime_top high watermark of any given service subclass with that of other subclasses affected by the same remapping threshold or thresholds. This is because an activity will complete after it has been remapped to a different service subclass by a remapping threshold, and the time the activity spends in other service subclasses before being remapped is counted only toward the service class in which it completes.

**Table 143. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return workload statistics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS</td>
<td>COLLECT AGGREGATE ACTIVITY DATA</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
</tbody>
</table>
Table 143. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function</td>
<td>COLLECT AGGREGATE ACTIVITY DATA</td>
</tr>
</tbody>
</table>

Table 144. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

This element can be used to help determine whether or not thresholds on activity lifetime are being effective and can also help to determine how to configure such thresholds.

**coord_act_queue_time_avg - Coordinator activity queue time average monitor element**

Arithmetic mean of queue time for coordinator activities at nesting level 0 associated with this service subclass or work class since the last reset.

If the internally tracked average has overflowed, the value -2 is returned. For service subclasses, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service subclass is set to NONE. For work classes, this monitor elements returns -1 if no COLLECT AGGREGATE ACTIVITY DATA work action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE. It is measured in milliseconds.

For service classes, the queue time counts only toward the service subclass in which the activity completes or is aborted. When you remap activities between service subclasses with a REMAP ACTIVITY action, the coord_act_queue_time_avg mean of service subclasses an activity is mapped to but does not complete in is unaffected.

Table 145. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function</td>
<td>COLLECT AGGREGATE ACTIVITY DATA</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function</td>
<td>COLLECT AGGREGATE ACTIVITY DATA</td>
</tr>
</tbody>
</table>
### Usage

Use this statistic to determine the arithmetic mean of the queue time for coordinator activities associated with a service subclass, workload, or work class that completed or aborted.

This statistic can also be used to determine whether or not the histogram template used for the activity queue time histogram is appropriate. Compute the average activity queue time from the activity queue time histogram. Compare the computed average with this monitor element. If the computed average deviates from the true average reported by this monitor element, consider altering the histogram template for the activity queue time histogram, using a set of bin values that are more appropriate for your data.

#### coord_act_rejected_total - Coordinator activities rejected total monitor element

The coord_act_rejected_total stores the total number of coordinator activities at any nesting level that were rejected instead of being allowed to execute since the last reset.

This counter is updated when an activity is prevented from executing by either a predictive threshold or a prevent execution work action. For service classes, the value is updated when the activity completes. For workloads, the value is updated by each workload occurrence at the end of its unit of work.

### Table 147. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

### Table 148. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
### Table 148. Event Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

#### Usage

This element can be used to help determine whether or not predictive thresholds and work actions that prevent execution are being effective and whether or not they are too restrictive.

**coord_partition_num - Coordinator partition number monitor element**

The coordinator partition of the unit of work or activity. In a multi-partition system, the coordinator partition is the partition where the application connected to the database.

#### Table 149. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

#### Table 150. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit of work</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

#### Usage

This element allows the coordinator partition to be identified for activities or units of work that have records on partitions other than the coordinator.

**cost_estimate_top - Cost estimate top monitor element**

The cost_estimate_top monitor element is a high watermark for the estimated cost of DML activities at all nesting levels in a service subclass or work class.

For service subclasses, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service subclass is set to NONE. For work classes, this monitor elements returns -1 if no COLLECT AGGREGATE ACTIVITY DATA work action is specified for the work class.

For service classes, the estimated cost of DML activities is counted only toward the service subclass in which the activity enters the system. When you remap activities between service subclasses with a REMAP ACTIVITY action, the cost_estimate_top of the service subclass you remap an activity to is unaffected.
Table 151. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine the highest DML activity estimated cost reached on a member for a service class, workload, or work class in the time interval collected.

**cpu_limit - WLM dispatcher CPU limit monitor element**

The WLM dispatcher CPU limit configured for the service class.

Table 152. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS</td>
<td>Always collected</td>
</tr>
<tr>
<td>- Get sample service class metrics</td>
<td></td>
</tr>
</tbody>
</table>

**cpu_share_type - WLM dispatcher CPU share type monitor element**

The type of WLM dispatcher CPU shares configured for the service class. Possible values are soft and hard.

Table 153. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS</td>
<td>Always collected</td>
</tr>
<tr>
<td>- Get sample service class metrics</td>
<td></td>
</tr>
</tbody>
</table>

**cpu_shares - WLM dispatcher CPU shares monitor element**

The number of WLM dispatcher CPU shares configured for the service class.

Table 154. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS</td>
<td>Always collected</td>
</tr>
<tr>
<td>- Get sample service class metrics</td>
<td></td>
</tr>
</tbody>
</table>

**cpu_utilization - CPU utilization monitor element**

The total CPU time consumed by the service class or workload on a particular logical partition divided by the amount of CPU time available on the host or the LPAR in a given period of time.

Table 155. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
</tbody>
</table>
Table 156. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_sscstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the system_metrics document</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>

Usage

When returned by the WLM_GET_WORKLOAD_STATS or the WLM_GET_SERVICE_SUBCLASS_STATS function, this monitor element represents the CPU utilization since the last reset of the statistics.

When returned by the MON_SAMPLE_SERVICE_CLASS_METRICS or the MON_SAMPLE_WORKLOAD_METRICS function, this monitor element represents the CPU utilization since the function was executed.

**cpu_velocity - CPU velocity monitor element**

A measure of the amount of contention for the CPU resources, measured on a scale from 0 to 1, with lower numbers meaning greater contention.

CPU velocity is computed by measuring the amount of time that work in a service class has access to the CPU divided by the total time spent accessing the CPU or waiting to access the CPU. It gives a measure of how efficiently the work is being executed relative to how efficiently it could be executed if such work never had to wait for the CPU. The formula is as follows:

\[
\text{cpu_velocity} = \frac{\text{total_cpu_time}}{\text{total_cpu_time} + \text{total_disp_run_queue_time}}
\]

The wlm_dispatcher database manager configuration parameter must be set to ON for cpu_velocity to be collected.

Table 157. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Sample service class metrics</td>
<td></td>
</tr>
</tbody>
</table>
Table 157. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_WORKLOAD_METRICS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Sample workload metrics</td>
<td></td>
</tr>
</tbody>
</table>

Usage

The dispatcher is effective at prioritizing a service class or workload when that service class or workload demands more CPU resources at a given instant than can be supplied. In such instances, the work executing in the service class or workload spends time queuing to access the CPU resources. It is when this occurs that the dispatcher can give more of the CPU resources to such a service class or workload by reducing how much of the CPU resources it gives to another. A high CPU velocity indicates that the dispatcher can have little effect on improving response times or throughput for this service class at its current level of CPU demand because this demand is already being met. A low CPU velocity indicates that the dispatcher can potentially have a significant effect on improving response times or throughput for this service class or workload at its current level of CPU demand.

Use this element to determine whether the work executing in a service class or workload is spending a relatively large proportion of its time queuing to use the CPU resources. If the CPU velocity for a service class is low and you want to increase it, you can adjust the WLM dispatcher control of the CPU resources by increasing the number of CPU shares or increasing the CPU limit assigned to the service class that is exhibiting a low CPU velocity.

**db_work_action_set_id - Database work action set ID monitor element**

If this activity has been categorized into a work class of database scope, this monitor element shows the ID of the work action set associated with the work class set to which the work class belongs. Otherwise, this monitor element shows the value of 0.

Table 158. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Command and Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 159. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

This element can be used with the **db_work_class_id** element to uniquely identify the database work class of the activity, if one exists.
**db_work_class_id - Database work class ID monitor element**

If this activity has been categorized into a work class of database scope, this monitor element displays the ID of the work class. Otherwise, this monitor element displays the value of 0.

**Table 160. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 161. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

This element can be used with the `db_work_action_set_id` element to uniquely identify the database work class of the activity, if one exists.

**destination_service_class_id - Destination service class ID monitor element**

The ID of the service subclass to which an activity was remapped when the threshold violation record to which this element belongs was generated. This element has a value of zero for any threshold action other than REMAP ACTIVITY.

**Table 162. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to trace the path of an activity through the service classes to which it was remapped. This element can also be used to compute aggregates of how many activities were mapped into a given service subclass.

**estimated_cpu_entitlement - Estimated CPU entitlement monitor element**

The percentage of total CPU on a host or an LPAR that a service subclass is configured to consume based on its CPU shares assuming that it consumes no more and no less than what it is configured to consume.

The determination of which service classes participate in its calculation is based on the actual CPU utilization measured over the sampling period versus the `WLM_DISP_MIN_UTIL` database manager configuration setting. The impact of a
CPU limit on a service class itself, on the service classes with which it competes, or on a parent service class (if it has one) are not taken into account in the calculation.

**Table 163. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

- Get sample service class metrics

**histogram_type - Histogram type monitor element**

The type of the histogram, in string format.

There are seven histogram types.

**CoordActQueueTime**
A histogram of the time (in milliseconds) non-nested activities spend queued (for example, in a threshold queue), measured on the coordinator member.

**CoordActExecTime**
A histogram of the time (in milliseconds) non-nested activities spend executing at the coordinator member. Execution time does not include time spent initializing or queued. For cursors, execution time includes only the time spent on open, fetch and close requests. When an activity is remapped between service subclasses, the execution time histogram is updated only for the service subclass in which the activity completes execution.

**CoordActLifetime**
A histogram of the elapsed time (in milliseconds) from when a non-nested activity is identified by the database manager until the activity completes execution, as measured on the coordinator member. When you remap activities between service subclasses, the lifetime histogram is updated only for the service subclass in which the activity completes execution.

**CoordActInterArrivalTime**
A histogram of the time interval (in milliseconds) between the arrival of non-nested coordinator activities. The inter-arrival time mean is calculated for service subclasses through which activities enter the system. When you remap activities between service subclasses, the inter-arrival time histogram of the service subclass you remap an activity to is unaffected.

**CoordActEstCost**
A histogram of the estimated cost (in timerons) of non-nested DML activities. The estimated cost of an activity is counted only toward the service subclass in which the activity enters the system.

**ReqExecTime**
A histogram of request execution times (in milliseconds), which includes requests on the coordinator member, and any subrequests on both coordinator and non-coordinator members (like RPC requests or SMP subagent requests). Requests included may or may not be associated with an activity: Both PREPARE and OPEN requests are included in this histogram, for example, but while OPEN requests are always associated with a cursor activity, PREPARE requests are not part of any activity. The execution time histogram of a service subclass involved in remapping counts the portion of the execution time spent by the partial request in the service subclass.
UowLifetime

A histogram of the elapsed time (in milliseconds) from the time that a unit of work is identified by the database manager until the time that the unit of work completes execution (committed or rolled back).

Table 164. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

Use this element to identify the type of histogram. Several histograms can belong to the same statistics record, but only one of each type.

last_wlm_reset - Time of last reset monitor element

This element, in the form of a local timestamp, shows the time at which the last statistics event record of this type was created.

Table 165. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_osmetrics</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scmetrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scmetrics</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlmetrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstat</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

Use the wlm_last_reset and statistics_timestamp monitor elements to determine a period of time over which the statistics in an event monitor statistics record were collected. The collection interval begins at the wlm_last_reset time and ends at statistics_timestamp.

num_remaps - Number of remaps monitor element

Count of the number of times this activity has been remapped. If num_remaps is greater than zero, the service_class_id of this activity record is the ID of the last service class to which the activity was remapped.

Table 166. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

Use this information to verify whether the activity was remapped the expected number of times.
num_threshold_violations - Number of threshold violations
monitor element

The number of threshold violations that have taken place in this database since it was last activated.

This monitor element is an alias of the “thresh_violations - Number of threshold violations monitor element” on page 486 monitor element, which is returned by some monitoring (MON_*) table functions.

Table 167. Snapshot Monitoring Information

<table>
<thead>
<tr>
<th>Snapshot Level</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>dbase</td>
<td>Basic</td>
</tr>
</tbody>
</table>

For snapshot monitoring, this counter can be reset.

Table 168. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>event_db</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

This element can be used to help determine whether or not thresholds are effective for this particular application or whether the threshold violations are excessive.

number_in_bin - Number in bin monitor element

This element holds the count of the number of activities or requests that fall within the histogram bin.

Table 169. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

Use this element to represent the height of a bin in the histogram.

parent_activity_id - Parent activity ID monitor element

The unique ID of the activity’s parent activity within the parent activity's unit of work. If there is no parent activity, the value of this monitor element is NULL.

Table 170. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Command and Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Table 170. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Command and Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return a list of activities</td>
<td></td>
</tr>
</tbody>
</table>

Table 171. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

Use this element along with the parent_uow_id element and appl_id element to uniquely identify the parent activity of the activity described in this activity record.

parent_uow_id - Parent unit of work ID monitor element

The unique unit of work identifier within an application handle. The ID of the unit of work in which the activity’s parent activity originates. If there is no parent activity, the value is NULL.

Table 172. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Command and Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return a list of activities</td>
<td></td>
</tr>
</tbody>
</table>

Table 173. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

Use this element along with the parent_activity_id element and appl_id element to uniquely identify the parent activity of the activity described in this activity record.

prep_time - Preparation time monitor element

Time in milliseconds required to prepare an SQL statement (if the activity is an SQL statement; otherwise, the value is 0).
Table 174. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT_DETAILS table function</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 175. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
<tr>
<td>Package cache</td>
<td></td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

The prep_time monitor element indicates how much time was spent preparing the SQL statement, if this activity was an SQL statement, when the statement was first introduced to the DB2 package cache. This preparation time is not part of the activity lifetime nor does it represent time spent during a specific invocation of the statement if the statement has already been cached in the package cache before that invocation.

queue_assignments_total - Queue assignments total monitor element

The number of times any connection or activity was assigned to this threshold queue since the last reset.

Table 176. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 177. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

This element can be used to determine the number of times any connection or activity was queued in this particular queue in a given period of time determined by the statistics collection interval. This can help to determine the effectiveness of queuing thresholds.
queue_size_top - Queue size top monitor element

Highest queue size that has been reached since the last reset.

Table 178. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function -</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return threshold queue statistics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function -</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return threshold queue statistics</td>
<td></td>
</tr>
</tbody>
</table>

Table 179. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

Use this element to gauge the effectiveness of queuing thresholds and to detect when queuing is excessive.

queue_time_total - Queue time total monitor element

Sum of the times spent in the queue for all connections or activities placed in this queue since the last reset. Units are milliseconds.

Table 180. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function -</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return threshold queue statistics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function -</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return threshold queue statistics</td>
<td></td>
</tr>
</tbody>
</table>

Table 181. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

This element is used to gauge the effectiveness of queuing thresholds and to detect when queuing is excessive.

Usage notes

queue_time_total is not reset at the end of a statistic collection interval. If queue_time_total is used over multiple intervals, it can be greater than the product of wlm_collect_int and queue_size_top.

request_exec_time_avg - Request execution time average monitor element

Arithmetic mean of the execution times for requests associated with this service subclass since the last reset. If the internally tracked average has overflowed, the
value -2 is returned. This monitor element returns -1 when COLLECT
AGGREGATE REQUEST DATA for the service subclass is set to NONE. Units are
milliseconds.

When you remap activities between service subclasses with a REMAP ACTIVITY
action, the request_exec_time_avg mean counts the partial request in each subclass
involved in remapping.

Table 182. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS</td>
<td>COLLECT AGGREGATE REQUEST DATA</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
</tbody>
</table>

Table 183. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scsstats</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

Use this statistic to quickly understand the average amount of time that is spent
processing each request on a member in this service subclass.

This average can also be used to determine whether or not the histogram template
used for the request execution time histogram is appropriate. Compute the average
request execution time from the request execution time histogram. Compare the
computed average with this monitor element. If the computed average deviates
from the true average reported by this monitor element, consider altering the
histogram template for the request execution time histogram, using a set of bin
values that are more appropriate for your data.

routine_id - Routine ID monitor element

A unique routine identifier. This monitor element returns 0 if the activity is not
part of a routine.

Table 184. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in the DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT_DETAILS - Get detailed metrics for package cache entries</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Table 184. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ROUTINE table function - get aggregated execution metrics for routines</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ROUTINE_DETAILS table function - get aggregated execution metric details for routines</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ROUTINE_EXEC_LIST table function - get list of statements executed by routine</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SECTION_ROUTINE table function - get list of routines for input section</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 185. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activitystmt</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>uow_package_list</td>
<td>Always collected</td>
</tr>
<tr>
<td>Package cache</td>
<td>pkgcache_metrics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

The value of this element matches a value from column ROUTINEID of view SYSCAT.ROUTINES. When the activity is part of an SQL PL routine that you declare in another SQL PL routine, the value of this element is the ROUTINEID of the outer routine.

rows_fetched - Rows fetched monitor element

The number of rows read from the table.

This monitor element is an alias of the rows_read monitor element.

Note: This monitor element reports only the values for the member for which this information is recorded. In multimember database environments, these values might not reflect the correct totals for the whole activity.

Table 186. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Statement</td>
</tr>
</tbody>
</table>

Usage

See the rows_read monitor element for details.

rows_modified - Rows modified monitor element

The number of rows inserted, updated, or deleted.

This monitor element is an alias of the rows_written monitor element.
### Table 187. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_FORMAT_XML_METRICS_BY_ROW - Get formatted row-based output for all metrics</td>
<td>Not applicable; reports whichever elements are in the XML document that is provided as input to the formatting function.</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTION_DETAILS table function - Get detailed connection metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASE table function - Get database level information</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASE_DETAILS table function - Get database information metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function - Get SQL statement activity metrics in the package cache</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT_DETAILS table function - get detailed metrics for package cache entries</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ROUTINE - get aggregated execution metrics for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ROUTINE_DETAILS - get aggregated execution metric details for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_DETAILS table function - Get detailed service subclass metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOADDETAILS table function - Get detailed workload metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>

### Table 188. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>
### Table 188. Event Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_sctstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the system_metrics document.</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Statement</td>
</tr>
<tr>
<td>Package cache</td>
<td>Reported in the activity_metrics document.</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>

#### Usage

See the **rows_written** monitor element for details.

**rows_returned - Rows returned monitor element**

The rows_returned monitor element is the number of rows that have been selected and returned to the application.

This element has a value of 0 for partial activity records (for example, if an activity is collected while it is still executing or when a full activity record could not be written to the event monitor due to memory limitations).

This monitor element is an alias of the **fetch_count** monitor element.

### Table 189. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_FORMAT_XML_METRICS_BY_ROW - Get formatted row-based output for all metrics</td>
<td>Not applicable; reports whichever elements are in the XML document that is provided as input to the formatting function.</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTIONDETAILS table function - Get detailed connection metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASE table function - Get database level information</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASEDETAILS table function - Get database information metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function - Get SQL statement activity metrics in the package cache</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMTDETAILS table function - get detailed metrics for package cache entries</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>
### Table 189. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ROUTINE - get aggregated execution metrics for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ROUTINE_DETAILS - get aggregated execution metric details for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_DETAILS table function - Get detailed service subclass metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOADDETAILS table function - Get detailed workload metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>

### Table 190. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Package cache</td>
<td>Reported in the activity_metrics document.</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the system_metrics document.</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>

### Usage

This element can be used to help determine thresholds for rows returned to the application or can be used to verify that such a threshold is configured correctly and doing its job.

**rows_returned_top - Actual rows returned top monitor element**

The rows_returned_top monitor element is the high watermark for the actual rows returned of DML activities at all nesting levels in a service class or work class.

For service classes, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class is set to NONE. For work classes, this
monitor element returns -1 if no COLLECT AGGREGATE ACTIVITY DATA work action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE.

For service classes, when you remap activities between service subclasses with a REMAP ACTIVITY action, only the rows_returned_top high watermark of the service subclass where an activity completes is updated. High watermarks of service subclasses an activity is mapped to but does not complete in are unaffected.

Table 191. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to know the highest DML activity actual rows returned reached on a member for a service class, workload, or work class in the time interval collected.

**sc_work_action_set_id - Service class work action set ID monitor element**

If this activity has been categorized into a work class of service class scope, this monitor element displays the ID of the work action set associated with the work class set to which the work class belongs. Otherwise, this monitor element displays the value of 0.

Table 192. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Command and Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 193. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

This element can be used with the **sc_work_class_id** element to uniquely identify the service class work class of the activity, if one exists.
**sc_work_class_id - Service class work class ID monitor element**

If this activity has been categorized into a work class of service class scope, this monitor element displays the ID of the work class assigned to this activity. Otherwise, this monitor element displays the value of 0.

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Command and Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 195. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activitystmt</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

This element can be used with the **sc_work_action_set_id** element to uniquely identify the service class work class of the activity, if one exists.

**section_env - Section environment monitor element**

A blob that contains the section for an SQL statement. It is the actual section contents, that is the executable form of the query plan.

<table>
<thead>
<tr>
<th>Table function</th>
<th>Monitor element collection level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SECTION table function - Get a copy of a section from the package cache</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 197. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activitystmt</td>
<td>Always collected</td>
</tr>
<tr>
<td>Package cache</td>
<td>pkgcache</td>
<td>COLLECT DETAILED DATA</td>
</tr>
</tbody>
</table>

**Usage**

Use this element with the section explain procedures to explain the statement and view the access plan for the statement.

**service_class_id - Service class ID monitor element**

Unique ID of service subclass. For a unit of work, this ID represents the service subclass ID of the workload with which the connection issuing the unit of work is associated.
### Table 198. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_DETAILS table function - Get detailed service subclass metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS - Get sample service class metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

### Table 199. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Locking</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

#### Usage

The value of this element matches a value from column SERVICECLASSID of view SYSCAT.SERVICECLASS. Use this element to look up the service subclass name, or link information about a service subclass from different sources. For example, join service class statistics with histogram bin records.

The value of this element is 0 when the following conditions are met:
- The element is reported in an event_histogrambin logical data group.
- The histogram data is collected for an object that is not a service class.
service_subclass_name - Service subclass name monitor element

The name of a service subclass.

**Table 200. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_DETAILS table function - Get detailed service subclass metrics (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORKDETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS - Get sample service class metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 201. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scsstats (reported in the details_xml document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Locking</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scsstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Usage

Use this element in conjunction with other activity elements for analysis of the behavior of an activity or with other statistics elements for analysis of a service class or threshold queue.

*service superclass name* - Service superclass name monitor element

The name of a service superclass.

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_DETAILS table function - Get detailed service subclass metrics (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUPERCLASS_STATS table function - Return statistics of service superclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORKDETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS table function - Get sample service class metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GETSERVICE_SUPERCLASS_STATS table function - Return statistics of service superclasses</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Table 203. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats (reported in the details_xml document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other activity elements for analysis of the behavior of an activity or with other statistics elements for analysis of a service class or threshold queue.

**source_service_class_id** - Source service class ID monitor element

The ID of the service subclass from which an activity was remapped when the threshold violation record to which this element belongs was generated. This element has a value of zero when the threshold action is anything other than a REMAP ACTIVITY action.

Table 204. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to trace the path of an activity through the service classes to which it was remapped. It can also be used to compute aggregates of how many activities were mapped out of a given service subclass.

**statistics_timestamp** - Statistics timestamp monitor element

The time at which this statistics record was generated.

Table 205. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_osmetrics</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to determine when this statistics record was generated.
Use this element along with the last_wlm_reset element to identify the time interval over which the statistics in this statistics record were generated.

This monitor element can also be used to group together all statistics records that were generated for the same collection interval.

**stmt_invocation_id - Statement invocation identifier monitor element**

An identifier that distinguishes one invocation of a routine from others at the same nesting level within a unit of work. It is unique within a unit of work for a specific nesting level.

**Table 206. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 207. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activitystmt</td>
<td>Always collected</td>
</tr>
<tr>
<td>Locking</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Deadlocks with Details History Values(^1)</td>
<td>event_stmt_history</td>
<td>Always collected</td>
</tr>
<tr>
<td>Deadlocks with Details History(^1)</td>
<td>event_stmt_history</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the package list.</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

\(^1\) This option has been 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.

**Usage**

You can use this element to uniquely identify the invocation in which a particular SQL statement has been executed. You can also use this element in conjunction with other statement history entries to see the sequence of SQL statements that caused the deadlock.

**temp_tablespace_top - Temporary table space top monitor element**

The temp_tablespace_top monitor element is a high watermark, in KB, for the temporary table space usage of DML activities at all nesting levels in a service class or work class.

For service classes, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class is set to NONE. For work classes, this monitor elements returns -1 if no COLLECT AGGREGATE ACTIVITY DATA work
action is specified for the work class. For workloads, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE.

For service classes, when you remap activities between service subclasses with a REMAP ACTIVITY action, only the temp_tablespace_top high watermark of the service subclass where an activity completes is changed. High watermarks of service subclasses an activity is mapped to but does not complete in are unaffected.

Table 208. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>-</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

Use this element to determine the highest DML activity system temporary table space usage reached on a member for a service class, workload, or work class in the time interval collected.

This element is only updated by activities that have a temporary table space threshold applied to them. If no temporary table space threshold is applied to an activity, a value of 0 is returned.

thresh Violations - Number of threshold violations monitor element

Number of times a threshold was violated.

This monitor element is an alias of the “num_threshold_violations - Number of threshold violations monitor element” on page 470 monitor element, which is returned by snapshot monitoring routines and the Database event monitor.

Table 209. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_FORMAT_XML_METRICS_BY_ROW - Get formatted row-based output for all metrics</td>
<td>Not applicable; reports whichever elements are in the XML document that is provided as input to the formatting function.</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTION_DETAILS table function - Get detailed connection metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASE table function - Get database level information</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>
**Table 209. Table Function Monitoring Information (continued)**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_DATABASE_DETAILS table function - Get</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>database information metrics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function - Get SQL</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>statement activity metrics in the package cache</td>
<td></td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT_DETAILS table function</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>- get detailed metrics for package cache entries</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ROUTINE - get aggregated execution metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>for routines table function</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ROUTINE_DETAILS - get aggregated</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>execution metric details for routines table function</td>
<td></td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>service subclass metrics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_DETAILS table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Get detailed service subclass metrics (reported</td>
<td></td>
</tr>
<tr>
<td>in DETAILS XML document)</td>
<td></td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>work metrics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Get detailed unit of work metrics (reported in</td>
<td></td>
</tr>
<tr>
<td>DETAILS XML document)</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>metrics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_DETAILS table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Get detailed workload metrics (reported in DETAILS</td>
<td></td>
</tr>
<tr>
<td>XML document)</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>- Return a list of activities</td>
<td></td>
</tr>
</tbody>
</table>

**Table 210. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scsstats (reported in the details_xml document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Package cache</td>
<td>Reported in the activity_metrics document.</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the system_metrics document.</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>

Chapter 8. Reference  487
Usage

Use this element to quickly determine if there have been any WLM thresholds that have been violated. If thresholds have been violated you can then use the threshold violations event monitor (if created and active) to obtain details about the threshold violations.

For example, to obtain details which threshold was violated.

**threshold_action - Threshold action monitor element**

The action of the threshold to which this threshold violation record applies.
Possible values include Stop, Continue and Remap.

*Table 211. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>-</td>
</tr>
</tbody>
</table>

Usage

Use this element to determine whether the activity that violated the threshold was stopped when the violation occurred, was allowed to continue executing, or was remapped to another service subclass. If the activity was stopped, the application that submitted the activity will have received an SQL4712N error. If the activity was remapped to another service subclass, agents working for the activity on the member will be moving to the target service subclass of the threshold.

**threshold_domain - Threshold domain monitor element**

The domain of the threshold responsible for this queue.

Possible values are
- Database
- Work Action Set
- Service Superclass
- Service Subclass
- Workload

*Table 212. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 213. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Usage

This element can be used for distinguishing the queue statistics of thresholds that have the same predicate but different domains.

**threshold_maxvalue - Threshold maximum value monitor element**

For non-queuing thresholds, this monitor element represents the value that was exceeded to cause this threshold violation. For queuing thresholds, this monitor element represents the level of concurrency that caused the queuing.

The level of concurrency that caused the violation of the queuing threshold is the sum of `threshold_maxvalue` and `threshold_queuesize` monitor elements.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

For activity thresholds, this element provides a historical record of what the threshold’s maximum value was at the time the threshold was violated. This is useful when the threshold’s maximum value has changed since the time of the violation and the old value is no longer available from the SYSCAT.THRESHOLDS view. For the DATATAGINSC IN and DATATAGINSC NOT IN thresholds, this element contains the value of the data tag that violated the threshold.

**threshold_name - Threshold name monitor element**

The unique name of the threshold responsible for this queue.

**Table 215. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element to uniquely identify the queuing threshold whose statistics this record represents.

**threshold_predicate - Threshold predicate monitor element**

Identifies the type of threshold that was violated or for which statistics were collected.
Table 217. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 218. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

Use this monitor element in conjunction with other statistics or threshold violation monitor elements for analysis of a threshold violation.

The valid values for this monitor element when reported in the `event_thresholdviolations` logical group are:

- AggSQLTempSpace
- SQLTempSpace
- SQLRowsReturned
- ActivityTotalTime
- EstimatedSQLCost
- TotalMemberConnections
- ConnectionIdleTime
- ConcurrentWorkloadOccurrences
- ConcurrentWorkloadActivities
- ConcurrentDBCoordActivities
- TotalSCMemberConnections
- SQLRowsRead
- SQLRowsReadInSC
- CPUtime
- CPUtimeInSC
- UowTotalTime
- DataTagInSC
- DataTagNotInSC

The valid values for this monitor element when reported in the `event_qstats` logical group are:

- TotalMemberConnections
- ConcurrentDBCoordActivities
threshold_queue_size - Threshold queue size monitor element

The size of the queue for a queuing threshold. An attempt to exceed this size causes a threshold violation. For a non-queuing threshold, this value is 0.

Table 219. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td></td>
</tr>
</tbody>
</table>

Usage

Use this element to determine the number of activities or connections in the queue for this threshold at the time the threshold was violated.

thresholdid - Threshold ID monitor element

Identifies the threshold to which a threshold violation record applies or for which queue statistics were collected.

Table 220. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td></td>
<td>Return threshold queue statistics</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td></td>
<td>Return threshold queue statistics</td>
</tr>
</tbody>
</table>

Table 221. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td></td>
</tr>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td></td>
</tr>
</tbody>
</table>

Usage

Use this monitor element in conjunction with other activity history monitor elements for analysis of a threshold queue or for analysis of the activity that violated a threshold.

time_completed - Time completed monitor element

The time at which the activity described by this activity record finished executing. This element is a local timestamp.

Table 222. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td></td>
</tr>
</tbody>
</table>

Usage

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity.
This field has a value of "0000-00-00-00.00.00.000000" when a full activity record could not be written to a table event monitor due to memory limitations. If the activity was captured while it was in progress, then this field represents the time that activity was collected.

**time_created - Time created monitor element**

The time at which a user submitted the activity described by this activity record. This element is a local timestamp.

Table 223. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity.

**time_of_violation - Time of violation monitor element**

The time at which the threshold violation described in this threshold violation record occurred. This element is a local timestamp.

Table 224. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other threshold violations monitor elements for analysis of a threshold violation.

**time_started - Time started monitor element**

The time at which the activity described by this activity record began executing. This element is a local timestamp.

Table 225. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity.

If the activity got rejected, then the value of **act_exec_time** monitor element is 0. In this case, the value of **time_started** monitor element equals the value of **time_completed** monitor element.

**top - Histogram bin top monitor element**

The inclusive top end of the range of a histogram bin. The value of this monitor element is also the bottom exclusive end of the range of the next histogram bin.
Table 226. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

Use this element with the corresponding `bottom` element to determine the range of a bin within a histogram.

**total_disp_run_queue_time - Total dispatcher run queue time monitor element**

The total time that requests, that were run in this service class, spent waiting to access the CPU. This value is given in microseconds.

Table 227. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_FORMAT_XML_METRICS_BY_ROW - Get formatted row-based output for all metrics</td>
<td>Not applicable; reports whichever elements are in the XML document that is provided as input to the formatting function.</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTIONDETAILS table function - Get detailed connection metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASE table function - Get database level information</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASEDETAILS table function - Get database information metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function - Get SQL statement activity metrics in the package cache</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMTDETAILS - Get detailed metrics for package cache entries</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ROUTINE - get aggregated execution metrics for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ROUTINEDETAILS - get aggregated execution metric details for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASSDETAILS table function - Get detailed service subclass metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>
Table 227. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOADDETAILS table function - Get detailed workload metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS - Get sample service class metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_SAMPLE_WORKLOAD_METRICS - Get sample workload metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function - Return statistics of service subclasses</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>

Table 228. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Package cache</td>
<td>Reported in the activity_metrics document</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_sctstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the system_metrics document</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>

Usage

Together with the total_cpu_time, the total_disp_run_queue_time monitor element can be used to compute a measure of the amount of contention for the CPU resource, measured on a scale from 0 to 1, with lower numbers meaning greater contention for the CPU resource. This measure, called CPU velocity, is computed by measuring the amount of time that work in a service class has access to the CPU divided by the total time spent accessing the CPU or waiting to access the CPU. It gives a measure of how efficiently the work is being executed relative to how efficiently it could be executed if such work never had to wait for the CPU. The formula is as follows:

\[ \text{CPU velocity} = \frac{\text{total_cpu_time}}{\text{total_cpu_time} + \text{total_disp_run_queue_time}} \]
When returned by the WLM_GET_SERVICE_SUBCLASS_STATS or the WLM_GET_WORKLOAD_STATS function, this monitor element represents the total dispatcher run queue wait time since the last reset of statistics.

When returned by the MON_SAMPLE_SERVICE_CLASS_METRICS or the MON_SAMPLE_WORKLOAD_METRICS function, this monitor element represents the total dispatcher run queue wait time since the function was executed.

**uow_completed_total - Total completed units of work monitor element**

The total number of units of work that completed, either by being committed or rolled back.

**Table 229. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>- Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>- Return workload statistics</td>
<td></td>
</tr>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS - Get sample</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- service class metrics</td>
<td></td>
</tr>
<tr>
<td>MON_SAMPLE_WORKLOAD_METRICS - Get sample workload</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- metrics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Return workload statistics</td>
<td></td>
</tr>
</tbody>
</table>

**Table 230. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats (reported in the</td>
<td>Always collected</td>
</tr>
<tr>
<td></td>
<td>metrics document)</td>
<td></td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the</td>
<td>Always collected</td>
</tr>
<tr>
<td></td>
<td>metrics document)</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

When returned by the WLM_GET_SERVICE_SUBCLASS_STATS or the WLM_GET_WORKLOAD_STATS function, this monitor element represents the total completed units of work since the last reset of the statistics.

When returned by the MON_SAMPLE_SERVICE_CLASS_METRICS or the MON_SAMPLE_WORKLOAD_METRICS function, this monitor element represents the total completed units of work since the function was executed.
**uow_comp_status - Unit of Work Completion Status**

The status of the unit of work and how it stopped.

**Table 231. Table function monitoring information**

<table>
<thead>
<tr>
<th>Table function</th>
<th>Monitor element collection level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_CONNECTION_DETAILS table function - Get detailed connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 232. Snapshot Monitoring Information**

<table>
<thead>
<tr>
<th>Snapshot Level</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>appl</td>
<td>Unit of Work</td>
</tr>
<tr>
<td>DCS Application</td>
<td>dcs_appl</td>
<td>Basic</td>
</tr>
</tbody>
</table>

**Table 233. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>event_xact</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage** You may use this element to determine if the unit of work ended due to a deadlock or abnormal termination. It may have been:
- Committed due to a commit statement
- Rolled back due to a rollback statement
- Rolled back due to a deadlock
- Rolled back due to an abnormal termination
- Committed at normal application termination.
- Unknown as a result of a FLUSH EVENT MONITOR command for which units of work were in progress.

**Note:** API users should refer to the header file (sqlmon.h) containing definitions of database system monitor constants.

**uow_elapsed_time - Most Recent Unit of Work Elapsed Time**

The elapsed execution time of the most recently completed unit of work.

**Element identifier**

uow_elapsed_time

**Element type**

time

**Table 234. Snapshot Monitoring Information**

<table>
<thead>
<tr>
<th>Snapshot Level</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>appl</td>
<td>Unit of Work, Timestamp</td>
</tr>
<tr>
<td>DCS Application</td>
<td>dcs_appl</td>
<td>Unit of Work, Timestamp</td>
</tr>
</tbody>
</table>
Usage

Use this element as an indicator of the time it takes for units of work to complete.

This element is composed of two subelements that report time spent as seconds and microseconds (one millionth of a second). The names of the subelements can be derived by adding ",_s" and ",_ms" to the name of this monitor element. To retrieve the total time spent for this monitor element, the values of the two subelements must be added together. For example, if the ",_s" subelement value is 3 and the ",_ms" subelement value is 20, then the total time spent for the monitor element is 3.00002 seconds.

**uow_id - Unit of work ID monitor element**

The unit of work identifier. The unit of work ID is unique within an application handle.

### Table 235. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

### Table 236. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitystmt</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activityvals</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>Change history</td>
<td>ddlstmtexec</td>
<td>Always collected</td>
</tr>
<tr>
<td></td>
<td>txncompletion</td>
<td></td>
</tr>
<tr>
<td>Locking</td>
<td>lock_participant_activities</td>
<td>Always collected</td>
</tr>
<tr>
<td></td>
<td>lock_activity_values</td>
<td></td>
</tr>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Table 236. Event Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit of work</td>
<td>uow</td>
<td>Always collected</td>
</tr>
<tr>
<td></td>
<td>uow_metrics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uow_package_list</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uow_executable_list</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity.

You can also use this element with the `activity_id` and `appl_id` monitor elements to uniquely identify an activity.

**uow_lifetime_avg - Unit of work lifetime average monitor element**

The average lifetime of a unit of work. Measured in milliseconds.

Table 237. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return workload statistics</td>
<td></td>
</tr>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Get sample service class metrics</td>
<td></td>
</tr>
<tr>
<td>MON_SAMPLE_WORKLOAD_METRICS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>- Get sample workload metrics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>table function - Return workload statistics</td>
<td></td>
</tr>
</tbody>
</table>

Table 238. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scsstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

When returned by the `WLM_GET_SERVICE_SUBCLASS_STATS` or the `WLM_GET_WORKLOAD_STATS` function, this monitor element represents the average unit of work lifetime since the last reset of the statistics.
When returned by the MON_SAMPLE_SERVICE_CLASS_METRICS or the MON_SAMPLE_WORKLOAD_METRICS function, this monitor element represents the average unit of work lifetime since the function was executed.

**uow_lock_wait_time - Total time unit of work waited on locks monitor element**

The total amount of elapsed time this unit of work has spent waiting for locks. The value is given in milliseconds.

**Element identifier**

uow_lock_wait_time

**Element type**

counter

**Table 239. Snapshot Monitoring Information**

<table>
<thead>
<tr>
<th>Snapshot Level</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>appl</td>
<td>Unit of Work</td>
</tr>
</tbody>
</table>

**Usage**

This element can help you determine the severity of the resource contention problem.

**uow_log_space_used - Unit of work log space used monitor element**

The amount of log space (in bytes) used in the current unit of work of the monitored application.

**Table 240. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 241. Snapshot Monitoring Information**

<table>
<thead>
<tr>
<th>Snapshot Level</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>appl</td>
<td>Unit of Work</td>
</tr>
</tbody>
</table>

**Table 242. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>event_xact</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>-</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

You may use this element to understand the logging requirements at the unit of work level.
**uow_start_time - Unit of work start timestamp monitor element**

The date and time that the unit of work first required database resources.

**Table 243. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_CONNECTION_DETAILS table function - Get detailed connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Table 244. Snapshot Monitoring Information**

<table>
<thead>
<tr>
<th>Snapshot Level</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>appl</td>
<td>Unit of Work, Timestamp</td>
</tr>
<tr>
<td>DCS Application</td>
<td>dcs_appl</td>
<td>Unit of Work, Timestamp</td>
</tr>
</tbody>
</table>

**Table 245. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit of work</td>
<td>uow</td>
<td>Always collected</td>
</tr>
<tr>
<td>Transaction</td>
<td>event_xact</td>
<td>-</td>
</tr>
</tbody>
</table>

**Usage**

This resource requirement occurs at the first SQL statement execution of that unit of work:

- For the first unit of work, it is the time of the first database request (SQL statement execution) after `conn_complete_time`.
- For subsequent units of work, it is the time of the first database request (SQL statement execution) after the previous COMMIT or ROLLBACK.

**Note:** The *SQL Reference* defines the boundaries of a unit of work as the COMMIT or ROLLBACK points.

The database system monitor excludes the time spent between the COMMIT/ROLLBACK and the next SQL statement from its definition of a unit of work. This measurement method reflects the time spent by the database manager in processing database requests, separate from time spent in application logic before the first SQL statement of that unit of work. The unit of work elapsed time does include the time spent running application logic between SQL statements within the unit of work.

You may use this element with the `uow_stop_time` monitor element to calculate the total elapsed time of the unit of work and with the `prev_uow_stop_time` monitor element to calculate the time spent in the application between units of work.
You can use the `uow_stop_time` and the `prev_uow_stop_time` monitor elements to calculate the elapsed time for the SQL Reference definition of a unit of work.

**uow_status - Unit of Work Status**

The status of the unit of work.

- **Element identifier**: `uow_status`
- **Element type**: information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>event_xact</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage** You may use this element to determine the status of a unit of work. API users should refer to the sqlmon.h header file containing definitions of database system monitor constants.

**uow_stop_time - Unit of work stop timestamp monitor element**

The date and time that the most recent unit of work completed, which occurs when database changes are committed or rolled back.

<table>
<thead>
<tr>
<th>Table function</th>
<th>Monitor element collection level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_CONNECTION_DETAILS table function - Get detailed connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage** Use this element with the `prev_uow_stop_time` monitor element to calculate the total elapsed time between COMMIT/ROLLBACK points, and with the `uow_start_time` monitor element to calculate the elapsed time of the latest unit of work.
The timestamp contents will be set as follows:

- When the application has completed a unit of work and has not yet started a new one (as defined in the `uow_start_time` monitor element), this element reports a valid, non-zero timestamp.
- When the application is currently executing a unit of work, this element reports zeros.
- When the application first connects to the database, this element is set to the value of the `conn_complete_time` monitor element.

As a new unit of work is started, the contents of this element are moved to the `prev_uow_stop_time` monitor element.

**uow_throughput - Unit of work throughput monitor element**

The completion rate of units of work measured in units of work per second.

**Table 250. Table Function Monitoring Information**

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_SAMPLE_SERVICE_CLASS_METRICS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Get sample service class metrics</td>
<td></td>
</tr>
<tr>
<td>MON_SAMPLE_WORKLOAD_METRICS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Get sample workload metrics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>table function - Return workload statistics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>table function - Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>table function - Return workload statistics</td>
<td></td>
</tr>
</tbody>
</table>

**Table 251. Event Monitoring Information**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_scstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

When returned by the `WLM_GET_SERVICE_SUBCLASS_STATS` or the `WLM_GET_WORKLOAD_STATS` function, this monitor element represents the unit of work throughput since the last reset of the statistics.

When returned by the `MON_SAMPLE_SERVICE_CLASS_METRICS` or the `MON_SAMPLE_WORKLOAD_METRICS` function, this monitor element represents the unit of work throughput since the function was executed.
uow_total_time_top - UOW total time top monitor element

High watermark for unit of work lifetime, in milliseconds.

Table 252. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_STATS table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return workload statistics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_SERVICE_SUBCLASS_STATS table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return statistics of service subclasses</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return workload statistics</td>
<td></td>
</tr>
</tbody>
</table>

Table 253. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Usage

This element can be used to help determine whether or not the UOWTOTALTIME threshold is effective and can also help to determine how to configure such a threshold.

For service classes, this monitor element returns -1 when COLLECT AGGREGATE ACTIVITY DATA for the service class is set to NONE.

For workloads, this monitor element returns -1 if COLLECT AGGREGATE ACTIVITY DATA for the workload is set to NONE.

For a service class, measurements taken for this high watermark are computed for the service class assigned by the workload. Any mapping by a work action set to change the service class of an activity does not affect this high watermark.

wl_work_action_set_id - Workload work action set identifier monitor element

If this activity has been categorized into a work class of workload scope, this monitor element shows the ID of the work action set associated with the work class set to which the work class belongs. Otherwise, this monitor element shows the value of 0.

Table 254. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
Table 254. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 255. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this monitor element, together with the `wl_work_class_id` monitor element, to uniquely identify the workload work class of the activity, if one exists.

**wl_work_class_id - Workload work class identifier monitor element**

If this activity has been categorized into a work class of workload scope, then this monitor element displays the identifier of the work class. Otherwise, this monitor element displays the value of 0.

Table 256. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY</td>
<td>Return a list of activities</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS</td>
<td>Get complete activity details (reported in DETAILS XML document)</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES</td>
<td>Return a list of activities</td>
</tr>
</tbody>
</table>

Table 257. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this monitor element, together with the `wl_work_action_set_id` monitor element, to uniquely identify the workload work class of the activity, if one exists.

**wlm_queue_assignments_total - Workload manager total queue assignments monitor element**

The number of times that activities or connections have been queued by a WLM threshold.

Table 258. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_FORMAT_XML_METRICS_BY_ROW</td>
<td>Get formatted row-based output for all metrics</td>
</tr>
</tbody>
</table>

Not applicable; reports whichever elements are in the XML document that is provided as input to the formatting function.
### Table 258. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ACTIVITYDETAILS table function - Get complete activity details (reported in DETAILS XML document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_CONNECTIONDETAILS table function - Get detailed connection metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASE table function - Get database level information</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_DATABASEDETAILS table function - Get database information metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function - Get SQL statement activity metrics in the package cache</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMTDETAILS table function - get detailed metrics for package cache entries</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ROUTINE - get aggregated execution metrics for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_ROUTINEDETAILS - get aggregated execution metric details for routines table function</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get service subclass metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASSDETAILS table function - Get detailed service subclass metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF WORKDETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOADDETAILS table function - Get detailed workload metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>

### Table 259. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_sctstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>
### Table 259. Event Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td></td>
<td>metrics document)</td>
<td></td>
</tr>
<tr>
<td>Package cache</td>
<td>Reported in the</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td></td>
<td>activity_metrics document.</td>
<td></td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td></td>
<td>system_metrics document.</td>
<td></td>
</tr>
</tbody>
</table>

### wlm_queue_time_total - Workload manager total queue time monitor element

The time spent waiting on a WLM queuing threshold. This value is given in milliseconds.

### Table 260. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_FORMAT_XML_METRICS_BY_ROW - Get formatted row-based</td>
<td>Not applicable; reports whichever elements are in the XML document that is provided as input to the</td>
</tr>
<tr>
<td>output for all metrics</td>
<td>formatting function.</td>
</tr>
<tr>
<td>MON_FORMAT_XML_TIMES_BY_ROW - Get formatted row-based</td>
<td>Not applicable; reports whichever elements are in the XML document that is provided as input to the</td>
</tr>
<tr>
<td>combined hierarchy wait and processing times</td>
<td>formatting function.</td>
</tr>
<tr>
<td>MON_FORMAT_XML_WAIT_TIMES_BY_ROW - Get formatted</td>
<td>Not applicable; reports whichever elements are in the XML document that is provided as input to the</td>
</tr>
<tr>
<td>row-based output for wait times</td>
<td>formatting function.</td>
</tr>
<tr>
<td>MON_GET_ACTIVITY table function - Return a list of</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>activities</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ACTIVITY_DETAILS table function - Get complete</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>activity details (reported in DETAILS XML document)</td>
<td></td>
</tr>
<tr>
<td>MON_GET_CONNECTION table function - Get connection</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>metrics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_CONNECTION_DETAILS table function - Get detailed</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>connection metrics (reported in DETAILS XML document)</td>
<td></td>
</tr>
<tr>
<td>MON_GET_DATABASE table function - Get database</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>level information</td>
<td></td>
</tr>
<tr>
<td>MON_GET_DATABASE_DETAILS table function - Get database</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>information metrics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT table function - Get SQL</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>statement activity metrics in the package cache</td>
<td></td>
</tr>
<tr>
<td>MON_GET_PKG_CACHE_STMT_DETAILS table function - get</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>detailed metrics for package cache entries</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ROUTINE - get aggregated execution metrics for</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>routines table function</td>
<td></td>
</tr>
<tr>
<td>MON_GET_ROUTINE_DETAILS - get aggregated execution</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>metric details for routines table function</td>
<td></td>
</tr>
<tr>
<td>MON_GET_SERVICE_SUBCLASS table function - Get</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>service subclass metrics</td>
<td></td>
</tr>
</tbody>
</table>
Table 260. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_SERVICE_SUBCLASS_DETAILS table function - Get detailed service subclass metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORKDETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload metrics</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>MON_GET_WORKLOADDETAILS table function - Get detailed workload metrics (reported in DETAILS XML document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function - Return a list of activities</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
</tbody>
</table>

Table 261. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activitymetrics</td>
<td></td>
</tr>
<tr>
<td>Statistics</td>
<td>event_sctstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>REQUEST METRICS BASE</td>
</tr>
<tr>
<td>Package cache</td>
<td>Reported in the activity_metrics document.</td>
<td>ACTIVITY METRICS BASE</td>
</tr>
<tr>
<td>Package cache</td>
<td>Reported in the system_metrics document.</td>
<td>REQUEST METRICS BASE</td>
</tr>
</tbody>
</table>

**wlo_completed_total - Workload occurrences completed total monitor element**

The number of workload occurrences to complete since last reset.

Table 262. Table Function Monitoring Information

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

Table 263. Event Monitoring Information

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
**Usage**

Use this element to determine how many occurrences of a given workload are driving work into the system.

**work_action_set_id - Work action set ID monitor element**

The ID of the work action set to which this statistics record applies.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**work_action_set_name - Work action set name monitor element**

The name of the work action set to which the statistics shown as part of this event are associated.

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return threshold queue statistics</td>
<td></td>
</tr>
<tr>
<td>MON_GET_WORK_ACTION_SET_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return work action set statistics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function</td>
<td>Always collected</td>
</tr>
<tr>
<td>Return threshold queue statistics</td>
<td></td>
</tr>
<tr>
<td>WLM_GET_WORK_ACTION_SET_STATS</td>
<td>Always collected</td>
</tr>
<tr>
<td>table function - Return work action set statistics</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity or with other statistics elements for analysis of a work class.

The value of this element is 0 when the following conditions are met:

- The element is reported in an event_histogrambin logical data group.
- The histogram data is collected for an object that is not a work class.

Use this element along with the `work_class_name` element to uniquely identify the work class whose statistics are being shown in this record or to uniquely identify
the work class which is the domain of the threshold queue whose statistics are shown in this record.

**work_class_id - Work class ID monitor element**

The identifier of the work class to which this statistics record applies.

*Table 267. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element in conjunction with other statistics elements for analysis of a work class.

The value of this element is 0 when the following conditions are met:
- The element is reported in an event_histogrambin logical data group.
- The histogram data is collected for an object that is not a work class.

**work_class_name - Work class name monitor element**

The name of the work class to which the statistics shown as part of this event are associated.

*Table 268. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORK_ACTION_SET_STATS table function - Return work action set statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORK_ACTION_SET_STATS table function - Return work action set statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 269. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>event_qstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wcstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this element along with the `work_action_set_name` element to uniquely identify the work class whose statistics are being shown in this record or to uniquely identify the work class which is the domain of the threshold queue whose statistics are shown in this record.
**workload_id - Workload ID monitor element**

An integer that uniquely identifies a workload.

*Table 270. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_DETAILS table function - Get detailed workload metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_SAMPLE_WORKLOAD_METRICS - Get sample</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 271. Snapshot Monitoring Information*

<table>
<thead>
<tr>
<th>Snapshot Level</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>appl_info</td>
<td>Basic</td>
</tr>
</tbody>
</table>

*Table 272. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locking</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_sccstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the system_metrics document</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_histogrambin</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity</td>
<td>Always collected</td>
</tr>
<tr>
<td>Threshold violations</td>
<td>event_thresholdviolations</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this ID to uniquely identify the workload to which this activity, application, histogram bin, or workload statistics record belongs.

The value of this element is 0 when the following conditions are met:
- The element is reported in an event_histogrambin logical data group.
- The histogram data is collected for an object that is not a workload.
**workload_name - Workload name monitor element**

Name of the workload.

*Table 273. Table Function Monitoring Information*

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_FORMAT_LOCK_NAME table function - Format the internal lock name and return details</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD table function - Get workload metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOADDETAILS table function - Get detailed workload metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_SAMPLE_WORKLOADMETRICS - Get sample</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_QUEUE_STATS table function - Return threshold queue statistics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_WORKLOAD_STATS table function - Return workload statistics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

*Table 274. Event Monitoring Information*

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Logical Data Grouping</th>
<th>Monitor Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locking</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>-</td>
<td>Always collected</td>
</tr>
<tr>
<td>Activities</td>
<td>event_activity (reported in the details_xml document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_scsstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats (reported in the metrics document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>Unit of work</td>
<td>Reported in the system_metrics document.</td>
<td>Always collected</td>
</tr>
<tr>
<td>Statistics</td>
<td>event_wlstats</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
**Usage**

In the statistics event monitor and workload table functions, the workload name identifies the workload for which statistics or metrics are being collected and reported. In the unit of work event monitor and unit of work table functions, the workload name identifies the workload that the unit of work was associated with.

Use the workload name to identify units of work or sets of information that apply to a particular workload of interest.

**workload_occurrence_id - Workload occurrence identifier**

monitor element

The ID of the workload occurrence to which this activity belongs.

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_AGENT table function - List agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORKDETAILS table function - Get detailed unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_AGENTS table function - list agents running in a service class</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_CONNECTIONDETAILS table function - Get detailed connection metrics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

**Usage**

Use this to identify the workload occurrence that submitted the activity.

**workload_occurrence_state - Workload occurrence state**

monitor element

The state of the workload occurrence.

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_CONNECTION table function - Get connection metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_CONNECTIONDETAILS table function - Get detailed connection metrics</td>
<td>Always collected</td>
</tr>
</tbody>
</table>
### Table 277. Table Function Monitoring Information (continued)

<table>
<thead>
<tr>
<th>Table Function</th>
<th>Monitor Element Collection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON_GET_UNIT_OF_WORK table function - Get unit of work metrics</td>
<td>Always collected</td>
</tr>
<tr>
<td>MON_GET_UNIT_OF_WORK_DETAILS table function - Get detailed unit of work metrics (reported in DETAILS XML document)</td>
<td>Always collected</td>
</tr>
<tr>
<td>WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function - list workload occurrences</td>
<td>Always collected</td>
</tr>
</tbody>
</table>

### Usage

Possible values include:

**DECOUPLED**
- Workload occurrence does not have a coordinator agent assigned (concentrator case).

**DISCONNECTPEND**
- Workload occurrence is disconnecting from the database.

**FORCED**
- Workload occurrence has been forced.

**INTERRUPTED**
- Workload occurrence has been interrupted.

**QUEUED**
- Workload occurrence coordinator agent is queued by a workload management queuing threshold. In a partitioned database environment, this state may indicate that the coordinator agent has made an RPC another member to obtain threshold tickets and has not yet received a response.

**TRANSIENT**
- Workload occurrence has not yet been mapped to a service superclass.

**UOWEXEC**
- Workload occurrence is processing a request.

**UOWWAIT**
- Workload occurrence is waiting for a request from the client.

### Commands

**SET WORKLOAD**

Specifies the workload to which the database connection is to be assigned.

This command can be issued before connecting to a database or it can be used to reassign the current connection once the connection has been established. If the connection has been established, the workload reassignment will be performed at the beginning of the next unit of work.
Authorization

None

Required connection

None

Command syntax

```
SET WORKLOAD TO AUTOMATIC
```

Command parameters

AUTOMATIC

Specifies that the database connection will be assigned to a workload chosen by the workload evaluation that is performed automatically by the server.

SYSDEFAULTADMWORKLOAD

Specifies that the database connection will be assigned to the SYSDEFAULTADMWORKLOAD, allowing users with accessctrl, dataaccess, wlmadm, secadm or dbadm authority to bypass the normal workload evaluation.

Examples

To assign the connection to the SYSDEFAULTADMWORKLOAD:

```
SET WORKLOAD TO SYSDEFAULTADMWORKLOAD
```

To reset the workload assignment so that it uses the workload that is chosen by the workload evaluation performed by the server:

```
SET WORKLOAD TO AUTOMATIC
```

Usage notes

If the session authorization ID of the database connection does not have accessctrl, dataaccess, wlmadm, secadm or dbadm authority, the connection cannot be assigned to the SYSDEFAULTADMWORKLOAD and an SQL0552N error will be returned. If the SET WORKLOAD TO SYSDEFAULTADMWORKLOAD command is issued before connecting to a database, the SQL0552N error will be returned after the database connection has been established, at the beginning of the first unit of work. If the command is issued when the database connection has been established, the SQL0552N error will be returned at the beginning of the next unit of work, when the workload reassignment is supposed to take place.

Configuration parameters

```
wlm_collect_int - Workload management collection interval configuration parameter
```

This parameter specifies a collect and reset interval, in minutes, for workload management (WLM) statistics.

Configuration type

Database
**Parameter type**
- Configurable online

**Default [range]**
- 0 [0 (no collection performed), 5 - 32767]

**Upgrade Note**
- If you are upgrading from a DB2 Version 9.8 Fix Pack 4 pureScale environment or earlier, the value of wlm_collect_int is set to the value on member 0.

Every \( x \) minutes, (where \( x \) is the value of the wlm_collect_int parameter) all workload management statistics are collected and sent to any active statistics event monitor; then the statistics are reset. If an active statistics event monitor exists, depending on how it was created, the statistics are written to a file, to a pipe, or to a table. If it does not exist, the statistics are only reset and not collected.

Collections occur at the specified interval times as measured relative to Sunday at 00:00:00. When the catalog member becomes active, the next collection occurs at the start of the next scheduled interval relative to this fixed time. The scheduled interval is not relative to the catalog member activation time. If a member is not active at the time of collection, no statistics are gathered for that member. For example, if the interval value was set to 60 and the catalog member was activated on 9:24 AM on Sunday, then the collections would be scheduled to occur each hour on the hour. Therefore, the next collection will occur at 10:00 AM. If the member is not active at 10:00 AM, then no statistics are gathered for that member.

The collect and reset process is initiated from the catalog member. The wlm_collect_int parameter must be specified on the catalog member. It is not used on other members.

The workload management statistics collected by a statistics event monitor can be used to monitor both short term and long term system behavior. A small interval can be used to obtain both short term and long term system behavior because the results can be merged together to obtain long term behavior. However, having to manually merge the results from different intervals complicates the analysis. If it is not required, a small interval unnecessarily increases the processing time. Therefore, reduce the interval to capture shorter term behavior, and increase the interval to reduce processing time when only analysis of long term behavior is sufficient.

The interval needs to be customized per database, not for each SQL request, or command invocation, or application. There are no other configuration parameters that need to be considered.

**Note:** All WLM statistics table functions return statistics that have been accumulated since the last time the statistics were reset. The statistics will be reset regularly on the interval specified by this configuration parameter.

**wlm_dispatcher - Workload management dispatcher**

This parameter enables (YES) or disables (NO) the DB2 workload management dispatcher. By default, an enabled dispatcher allows the setting of the CPU limits.

**Configuration type**
- Database manager

**Applies to**
- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**
Configurable Online

**Propagation class**
Immediate

**Default [range]**
NO [NO; YES]
When upgrading the DB2 database manager, the value of the `wlm_dispatcher` database manager configuration parameter is set to NO.

The workload management dispatcher provides CPU scheduling capabilities at the service class level in the DB2 database manager using shares-based allocation of CPU resources, or CPU limits, or both.

With the workload management dispatcher enabled, all work running in user and maintenance service classes is placed under the control of the dispatcher. When enabled, CPU limit settings are enforced by the dispatcher as the default case. In order to use shares-based allocation of CPU resources, the `wlm_disp_cpu_shares` database manager configuration parameter must be enabled.

**wlm_disp_concur - Workload manager dispatcher thread concurrency**

This parameter specifies how the DB2 workload manager (WLM) dispatcher sets the thread concurrency level. You can also manually set the thread concurrency level to a fixed value.

**Configuration type**
Database manager

**Applies to**
- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**
Configurable Online

**Propagation class**
Immediate

**Default [range]**
COMPUTED [COMPUTED; manually_set_value]

When upgrading DB2 database manager, the value of the `wlm_disp_concur` database manager configuration parameter is COMPUTED.

**COMPUTED**
DB2 database manager computes a fixed thread concurrency level based upon the value of 4 times the number of logical CPUs available to the DB2 database manager.

**manually_set_value**
You can manually set the thread concurrency level to a fixed value (1 - 32767). The optimal value depends on the specific hardware
used and the operating system level; generally, in the range of 2 to 4 times the number of logical CPUs on the host or LPAR.

**Unit of measure**

Number of concurrent threads

The setting of this database manager configuration parameter controls the number of threads that the WLM dispatcher allows to be dispatched to the operating system run queues in parallel. The value is set as a low multiple of the number of logical CPUs available to the DB2 database manager. In general, you can set the value to 4 times the number of available logical CPUs to take into account possible scheduling latencies that result when threads switch in and out of the active state. An optimal value is just large enough to ensure that there are adequate numbers of threads for the DB2 database manager to fully use the CPUs on the host or LPAR and no larger. This optimal value ensures maximum efficiency and gives the DB2 WLM dispatcher maximum control over CPU allocation.

**wlm_disp_cpu_shares - Workload manager dispatcher CPU shares**

This parameter enables (YES) or disables (NO) the control of CPU shares by the DB2 workload manager (WLM) dispatcher. By default, an enabled WLM dispatcher controls only CPU limits.

**Configuration type**

Database manager

**Applies to**

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

**Parameter type**

Configurable Online

**Propagation class**

Immediate

**Default [range]**

NO [NO; YES]

When upgrading DB2 database manager, the value of the `wlm_disp_cpu_shares` database manager configuration parameter is NO.

If the value of the `wlm_dispatcher` database manager configuration parameter is set to YES and the value of the `wlm Disp_cpu_shares` database manager configuration parameter is set to NO, the WLM dispatcher can apply only CPU limits to the management of service classes.

If the value of the `wlm_dispatcher` database manager configuration parameter is set to YES and the value of the `wlm Disp_cpu_shares` database manager configuration parameter is set to YES, the WLM dispatcher can apply both CPU limits and CPU shares to the management of service classes. By default, all service classes are assigned 1000 hard CPU shares to ensure an equal division of CPU resources.
### Table 278. Summary of required database manager configuration parameter settings for service class management by the DB2 WLM dispatcher

<table>
<thead>
<tr>
<th>Service class management</th>
<th>Setting of <code>wlm_disp_cpu_shares</code></th>
<th>Setting of <code>wlm_disp_cpu_shares</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>CPU limits</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>CPU limits + CPU shares</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

#### `wlm_disp_min_util` - Workload manager dispatcher minimum CPU utilization

This parameter specifies the minimum amount of CPU utilization that is necessary for a service class to be included in the DB2 WLM-managed sharing of CPU resources.

**Configuration type**
- Database manager

**Applies to**
- Database server with local and remote clients
- Database server with local clients
- Multimember database server with local and remote clients

**Parameter type**
- Configurable Online

**Propagation class**
- Immediate

**Default [range]**
- 5 [0 to 100]

When upgrading DB2 database manager, the value of the `wlm_disp_min_util` database manager configuration parameter is 5.

**Unit of measure**
- Percent

To illustrate the usage of this database manager configuration parameter with an example, suppose there are three service classes, A, B, and C, and each has 1000 shares of the CPU resources. In this example, the same result is obtained whether the service class shares are hard or soft CPU shares. Service classes A and B each have a CPU utilization that is greater than or equal to the 8% value set for the `wlm_disp_min_util` configuration parameter. Service class C has a 3% CPU utilization that is less than the 8% value set for the `wlm_disp_min_util` configuration parameter. In CPU share calculations, service class C is considered to not have any executing work. Therefore, only service classes A and B equally share the CPU resources and each receives a 50% share. When service class C begins to execute work to an extent that the CPU utilization is greater than or equal to the 8% value set for the `wlm Disp_Min_Util` configuration parameter, at this point service classes A, B, and C are now considered to equally share the CPU resources and each receives a 33.3% share.

In multimember database environments, it is the aggregate of the CPU utilizations of all the members on a host or LPAR that is compared to the `wlm_disp_min_util` configuration parameter to determine if the host or LPAR is included in the WLM-managed sharing of CPU resources.
**Catalog views**

**SYSCAT.HISTOGRAMTEMPLATEBINS**

Each row represents a histogram template bin.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPLATENAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the histogram template.</td>
</tr>
<tr>
<td>TEMPLATEID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the histogram template.</td>
</tr>
<tr>
<td>BINID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the histogram template bin.</td>
</tr>
<tr>
<td>BINUPPERVALUE</td>
<td>BIGINT</td>
<td></td>
<td>The upper value for a single bin in the histogram template.</td>
</tr>
</tbody>
</table>

**SYSCAT.HISTOGRAMTEMPLATES**

Each row represents a histogram template.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPLATEID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the histogram template.</td>
</tr>
<tr>
<td>TEMPLATENAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the histogram template.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the histogram template was created.</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the histogram template was last altered.</td>
</tr>
<tr>
<td>NUMBINS</td>
<td>INTEGER</td>
<td></td>
<td>Number of bins in the histogram template, including the last bin that has an unbounded top value.</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR (254)</td>
<td>Y</td>
<td>User-provided comments, or the null value.</td>
</tr>
</tbody>
</table>

**SYSCAT.HISTOGRAMTEMPLATEUSE**

Each row represents a relationship between a workload management object that can use histogram templates and a histogram template.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPLATENAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the histogram template.</td>
</tr>
<tr>
<td>TEMPLATEID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the histogram template.</td>
</tr>
<tr>
<td>HISTOGRAMTYPE</td>
<td>CHAR (1)</td>
<td></td>
<td>The type of information collected by histograms based on this template.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C = Activity estimated cost histogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E = Activity execution time histogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• I = Activity interarrival time histogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• L = Activity life time histogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Q = Activity queue time histogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• R = Request execution time histogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• U = Unit of work life time histogram</td>
</tr>
</tbody>
</table>
Table 281. SYSCAT.HISTOGRAMTEMPLATEUSE Catalog View (continued)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTTYPE</td>
<td>CHAR (1)</td>
<td></td>
<td>The type of WLM object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• b = Service class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• k = Work action</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• w = Workload</td>
</tr>
<tr>
<td>OBJECTID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier of the WLM object.</td>
</tr>
<tr>
<td>SERVICECLASSNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the service class.</td>
</tr>
<tr>
<td>PARENTSERVICECLASSNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>The name of the parent service class of the service subclass that uses the histogram template.</td>
</tr>
<tr>
<td>WORKACTIONNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>The name of the work action that uses the histogram template.</td>
</tr>
<tr>
<td>WORKACTIONSETNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>The name of the work action set containing the work action that uses the histogram template.</td>
</tr>
<tr>
<td>WORKLOADNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>The name of the workload that uses the histogram template.</td>
</tr>
</tbody>
</table>

SYSCAT.SERVICECLASSES

Each row represents a service class.

Table 282. SYSCAT.SERVICECLASSES Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICECLASSNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the service class.</td>
</tr>
<tr>
<td>PARENTSERVICECLASSNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Service class name of the parent service superclass.</td>
</tr>
<tr>
<td>SERVICECLASSID</td>
<td>SMALLINT</td>
<td></td>
<td>Identifier for the service class.</td>
</tr>
<tr>
<td>PARENTID</td>
<td>SMALLINT</td>
<td></td>
<td>Identifier for the parent service class for this service class. 0 if this service class is a super service class.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time when the service class was created.</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time when the service class was last altered.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>CHAR (1)</td>
<td></td>
<td>State of the service class.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = Enabled</td>
</tr>
<tr>
<td>AGENTPRIORITY</td>
<td>SMALLINT</td>
<td></td>
<td>This column is no longer used and will be removed in a future release.</td>
</tr>
<tr>
<td>PREFETCHPRIORITY</td>
<td>CHAR (1)</td>
<td></td>
<td>Prefetch priority of the agents in the service class.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H = High</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• L = Low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• M = Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Blank = not set</td>
</tr>
<tr>
<td>MAXDEGREE</td>
<td>SMALLINT</td>
<td>Y</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>
**Table 282. SYSCAT.SERVICECLASSES Catalog View (continued)**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFERPOOLPRIORITY</td>
<td>CHAR (1)</td>
<td></td>
<td>Bufferpool priority of the agents in the service class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H = High</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• L = Low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• M = Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Blank = Not set</td>
</tr>
<tr>
<td>INBOUNDCORRELATOR</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>For future use.</td>
</tr>
<tr>
<td>OUTBOUNDCORRELATOR</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>String used to associate the service class with an operating system workload manager service class.</td>
</tr>
<tr>
<td>COLLECTAGGACTDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what aggregate activity data should be captured for the service class by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base aggregate activity data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E = Collect extended aggregate activity data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td>COLLECTAGGREQDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what aggregate request data should be captured for the service class by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base aggregate request data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td>COLLECTACTDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what activity data should be collected by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = Activity data with details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S = Activity data with details and section environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• V = Activity data with details and values</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• W = Activity data without details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X = Activity data with details, section environment, and values</td>
</tr>
<tr>
<td>COLLECTACTPARTITION</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies where activity data is collected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C = Coordinator member of the activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = All members</td>
</tr>
<tr>
<td>COLLECTREQMETRICS</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies the monitoring level for requests submitted by a connection that is associated with the service superclass.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base request metrics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E = Collect extended request metrics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td>CPUSHARES</td>
<td>INTEGER</td>
<td></td>
<td>The number of CPU shares allocated to this service class.</td>
</tr>
<tr>
<td>CPUSHARETYPE</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies the type of CPU shares.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S = Soft shares</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H = Hard shares</td>
</tr>
</tbody>
</table>
### Table 282. SYSCAT.SERVICECLASSES Catalog View (continued)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPULIMIT</td>
<td>SMALLINT</td>
<td></td>
<td>The maximum percentage of the CPU resource that can be allocated to the service class; -1 if there is no CPU limit.</td>
</tr>
<tr>
<td>SORTMEMORYPRIORITY</td>
<td>CHAR (1)</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>SECTIONACTUALSOPTIONS</td>
<td>VARCHAR (32)</td>
<td></td>
<td>Specifies what section actuals are collected during the execution of a section. The first position in the string represents whether the collection of section actuals is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Enabled and collect basic operator cardinality counts and statistics for each object referenced by the section (DML statements only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Not enabled. The second position is always 'N' and reserved for future use.</td>
</tr>
<tr>
<td>COLLECTAGGUOWDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what aggregate unit of work data should be captured for the service class by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base aggregate unit of work data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR (254)</td>
<td>Y</td>
<td>User-provided comments, or the null value.</td>
</tr>
</tbody>
</table>

### SYSCAT.THRESHOLDS

Each row represents a threshold.

### Table 283. SYSCAT.THRESHOLDS Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRESHOLDNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the threshold.</td>
</tr>
<tr>
<td>THRESHOLDID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the threshold.</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>CHAR (1)</td>
<td></td>
<td>Origin of the threshold.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• U = Threshold was created by a user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• W = Threshold was created through a work action set</td>
</tr>
<tr>
<td>THRESHOLDCLASS</td>
<td>CHAR (1)</td>
<td></td>
<td>Classification of the threshold.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A = Aggregate threshold</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C = Activity threshold</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Nullable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>THRESHOLDPREDICATE</td>
<td>VARCHAR (15)</td>
<td></td>
<td>Type of the threshold. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- AGGTEMPSPACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CONCDBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CONCWCN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CONCWOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CONNIDLTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CPUTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CPUTIMEINSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DATAGINSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DATAGNOTINSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DBCONN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ESTSQLCOST</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ROWSREAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ROWSREADINSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ROWSRET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- SCCONN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- TEMPSPACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- TOTALTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- UOWTOTALTIME</td>
</tr>
<tr>
<td>THRESHOLDPREDICATEID</td>
<td>SMALLINT</td>
<td></td>
<td>Identifier for the threshold predicate.</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>CHAR (2)</td>
<td></td>
<td>Domain of the threshold.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DB = Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SB = Service subclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SP = Service superclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WA = Work action set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WD = Workload definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SQ = SQL statement</td>
</tr>
<tr>
<td>ENFORCEMENT</td>
<td>CHAR (1)</td>
<td></td>
<td>Scope of enforcement for the threshold.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D = Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>P = Member</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W = Workload occurrence</td>
</tr>
<tr>
<td>QUEUING</td>
<td>CHAR (1)</td>
<td></td>
<td>N = The threshold is not queueing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y = The threshold is queueing</td>
</tr>
<tr>
<td>MAXVALUE</td>
<td>BIGINT</td>
<td></td>
<td>Upper bound specified by the threshold. If THRESHOLDPREDICATE is 'DATAGINSC' or 'DATAGNOTINSC', this value encodes one or more data tags.</td>
</tr>
<tr>
<td>DATATAGLIST</td>
<td>VARCHAR (256)</td>
<td>Y</td>
<td>If THRESHOLDPREDICATE is 'DATAGINSC' or 'DATAGNOTINSC', this value represents one or more data tags as a comma separated list. Otherwise, the null value.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Nullable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QUEUESIZE</td>
<td>INTEGER</td>
<td></td>
<td>If QUEUEING is 'Y', the size of the queue. -1 otherwise.</td>
</tr>
<tr>
<td>OVERFLOWPERCENT</td>
<td>SMALLINT</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>COLLECTACTDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what activity data should be collected by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = Activity data with details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S = Activity data with details and section environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• V = Activity data with details and values</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• W = Activity data without details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X = Activity data with details, section environment, and values</td>
</tr>
<tr>
<td>COLLECTACTPARTITION</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies where activity data is collected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C = Coordinator member of the activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = All members</td>
</tr>
<tr>
<td>EXECUTION</td>
<td>CHAR (1)</td>
<td></td>
<td>Indicates the execution action taken after a threshold has been exceeded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C = Execution continues</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• F = Application is forced off the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• R = Execution is remapped to a different service subclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S = Execution stops</td>
</tr>
<tr>
<td>REMAPSCID</td>
<td>SMALLINT</td>
<td></td>
<td>Target service subclass ID of the REMAP ACTIVITY action.</td>
</tr>
<tr>
<td>VIOLATIONRECORDLOGGED</td>
<td>CHAR (1)</td>
<td></td>
<td>Indicates whether a record is written to the event monitor upon threshold violation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = Yes</td>
</tr>
<tr>
<td>CHECKINTERVAL</td>
<td>INTEGER</td>
<td></td>
<td>The interval, in seconds, in which the threshold condition is checked if THRESHOLDPREDICATE is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 'CPUTIME'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 'CPUTIMEINSC'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 'ROWSREAD'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 'ROWSREADINSC'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Otherwise, -1.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>CHAR (1)</td>
<td></td>
<td>• N = This threshold is disabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = This threshold is enabled.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the threshold was created.</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the threshold was last altered.</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR (254)</td>
<td>Y</td>
<td>User-provided comments, or the null value.</td>
</tr>
</tbody>
</table>
SYSCAT.WORKACTIONS
Each row represents a work action that is defined for a work action set.

Table 284. SYSCAT.WORKACTIONS Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the work action.</td>
</tr>
<tr>
<td>ACTIONID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the work action.</td>
</tr>
<tr>
<td>ACTIONSETNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the work action set.</td>
</tr>
<tr>
<td>ACTIONSETID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier of the work action set to which this work action belongs.</td>
</tr>
<tr>
<td>WORKCLASSNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the work class.</td>
</tr>
<tr>
<td>WORKCLASSID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier of the work class.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work action was created.</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work action was last altered.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>CHAR (1)</td>
<td></td>
<td>• N = This work action is disabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = This work action is enabled.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Nullable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>ACTIONTYPE</td>
<td>CHAR (1)</td>
<td></td>
<td>The type of action performed on each DB2 activity that matches the attributes in the work class within scope.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect basic aggregate activity data, specifiable only for work action sets that apply to service classes or workloads.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C = Allow any DB2 activity under the associated work class to execute and increment the work class counter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = Collect activity data with details at the coordinating member of the activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E = Collect extended aggregate activity data, specifiable only for work action sets that apply to service classes or workloads.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• F = Collect activity data with details, section, and values at the coordinating member of the activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• G = Collect activity details and section at the coordinating member of the activity and collect activity data at all members.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H = Collect activity details, section, and values at the coordinating member of the activity and collect activity data at all members.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• M = Map to a service subclass, specifiable only for work action sets that apply to service classes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• P = Prevent the execution of any DB2 activity under the work class with which this work action is associated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S = Collect activity data with details and section at the coordinating member of the activity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• T = The action represents a threshold, specifiable only for work action sets that are associated with a database or a workload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• U = Map all activities with a nesting level of zero and all activities nested under these activities to a service subclass, specifiable only for work action sets that apply to service classes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• V = Collect activity data with details and values at the coordinating member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• W = Collect activity data without details at the coordinating member.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X = Collect activity data with details at the coordinating member and collect activity data at all members.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = Collect activity data with details and values at the coordinating member and collect activity data at all members.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Z = Collect activity data without details at all members.</td>
</tr>
</tbody>
</table>
### Table 284. SYSCAT.WORKACTIONS Catalog View (continued)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFOBJECTID</td>
<td>INTEGER</td>
<td>Y</td>
<td>If ACTIONTYPE is 'M' (map) or 'N' (map nested), this value is set to the ID of the service subclass to which the DB2 activity is mapped. If ACTIONTYPE is 'T' (threshold), this value is set to the ID of the threshold to be used. For all other actions, this value is NULL.</td>
</tr>
<tr>
<td>REFOBJECTTYPE</td>
<td>VARCHAR (30)</td>
<td></td>
<td>If the ACTIONTYPE is 'M' or 'N', this value is set to 'SERVICE CLASS'; if the ACTIONTYPE is 'T', this value is 'THRESHOLD'; the null value otherwise.</td>
</tr>
<tr>
<td>SECTIONACTUALSOPTIONS</td>
<td>VARCHAR (32)</td>
<td></td>
<td>Specifies what section actuals are collected during the execution of a section. The first position in the string represents whether the collection of section actuals is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Enabled and collect basic operator cardinality counts and statistics for each object referenced by the section (DML statements only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Not enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The second position is always 'N' and reserved for future use.</td>
</tr>
</tbody>
</table>

### SYSCAT.WORKACTIONSETS

Each row represents a work action set.

### Table 285. SYSCAT.WORKACTIONSETS Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONSETNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the work action set.</td>
</tr>
<tr>
<td>ACTIONSETID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the work action set.</td>
</tr>
<tr>
<td>WORKCLASSSETNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the work class set.</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>INTEGER</td>
<td></td>
<td>The identifier of the work class set that is to be mapped to the object specified by the OBJECTID. This column refers to WORKCLASSSETID in the SYSCAT.WORKCLASSSETS view.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work action set was created.</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work action set was last altered.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>CHAR (1)</td>
<td></td>
<td>• N = This work action set is disabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = This work action set is enabled.</td>
</tr>
<tr>
<td>OBJECTTYPE</td>
<td>CHAR (1)</td>
<td></td>
<td>• b = Service superclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• w = Workload</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Blank = Database</td>
</tr>
<tr>
<td>OBJECTNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the service class or workload.</td>
</tr>
</tbody>
</table>
Table 285. SYSCAT.WORKACTIONSETS Catalog View (continued)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTID</td>
<td>INTEGER</td>
<td></td>
<td>The identifier of the object to which the work class set (specified by the WORKCLASSSETID) is mapped. If the OBJECTTYPE is 'b', the OBJECTID is the ID of the service superclass. If the OBJECTTYPE is 'w', the OBJECTID is the ID of the workload. If the OBJECTTYPE is blank, the OBJECTID is -1.</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR (254)</td>
<td>Y</td>
<td>User-provided comments, or the null value.</td>
</tr>
</tbody>
</table>

SYSCAT.WORKCLASSES
Each row represents a work class defined for a work class set.

Table 286. SYSCAT.WORKCLASSES Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKCLASSNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the work class.</td>
</tr>
<tr>
<td>WORKCLASSSETNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the work class set.</td>
</tr>
<tr>
<td>WORKCLASSID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the work class.</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the work class set to which this work class belongs. This column refers to the WORKCLASSSETID column in the SYSCAT.WORKCLASSSETS view.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work class was created.</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work class was last altered.</td>
</tr>
<tr>
<td>EVALUATIONORDER</td>
<td>SMALLINT</td>
<td></td>
<td>Uniquely identifies the evaluation order used for choosing a work class within a work class set.</td>
</tr>
</tbody>
</table>

SYSCAT.WORKCLASSSETS
Each row represents a work class set.

Table 287. SYSCAT.WORKCLASSSETS Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKCLASSSETNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the work class set.</td>
</tr>
<tr>
<td>WORKCLASSSETID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the work class set.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work class set was created.</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the work class set was last altered.</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR (254)</td>
<td>Y</td>
<td>User-provided comments, or the null value.</td>
</tr>
</tbody>
</table>

SYSCAT.WORKLOADAUTH
Each row represents a user, group, or role that has been granted USAGE privilege on a workload.
Table 288. SYSCAT.WORKLOADAUTH Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKLOADID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the workload.</td>
</tr>
<tr>
<td>WORKLOADNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the workload.</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Grantor of the privilege.</td>
</tr>
<tr>
<td>GRANTORTYPE</td>
<td>CHAR (1)</td>
<td></td>
<td>• U = Grantee is an individual user</td>
</tr>
<tr>
<td>GRANTEE</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Holder of the privilege.</td>
</tr>
<tr>
<td>GRANTEETYPE</td>
<td>CHAR (1)</td>
<td></td>
<td>• G = Grantee is a group</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• R = Grantee is a role</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• U = Grantee is an individual user</td>
</tr>
<tr>
<td>USAGEAUTH</td>
<td>CHAR (1)</td>
<td></td>
<td>Indicates whether grantee holds USAGE privilege on the workload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Not held</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = Held</td>
</tr>
</tbody>
</table>

SYSCAT.WORKLOADCONNATTR

Each row represents a connection attribute in the definition of a workload.

Table 289. SYSCAT.WORKLOADCONNATTR Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKLOADID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the workload.</td>
</tr>
<tr>
<td>WORKLOADNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the workload.</td>
</tr>
<tr>
<td>CONNATTRTYPE</td>
<td>VARCHAR (30)</td>
<td></td>
<td>Type of the connection attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1 = APPLNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 = SYSTEM_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3 = SESSION_USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 4 = SESSION_USER GROUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 5 = SESSION_USER ROLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 6 = CURRENT CLIENT_USERID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 7 = CURRENT CLIENT_APPLNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 8 = CURRENT CLIENT_WRKSTNNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 9 = CURRENT CLIENT_ACCTNG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 10 = ADDRESS</td>
</tr>
<tr>
<td>CONNATTRVALUE</td>
<td>VARCHAR (1000)</td>
<td></td>
<td>Value of the connection attribute.</td>
</tr>
</tbody>
</table>

SYSCAT.WORKLOADS

Each row represents a workload.

Table 290. SYSCAT.WORKLOADS Catalog View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Nullable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKLOADID</td>
<td>INTEGER</td>
<td></td>
<td>Identifier for the workload.</td>
</tr>
<tr>
<td>WORKLOADNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the workload.</td>
</tr>
<tr>
<td>EVALUATIONORDER</td>
<td>SMALLINT</td>
<td></td>
<td>Evaluation order used for choosing a workload.</td>
</tr>
<tr>
<td>CREATE_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the workload was created.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Nullable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ALTER_TIME</td>
<td>TIMESTAMP</td>
<td></td>
<td>Time at which the workload was last altered.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>CHAR (1)</td>
<td></td>
<td>• N = This workload is disabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = This workload is enabled.</td>
</tr>
<tr>
<td>ALLOWACCESS</td>
<td>CHAR (1)</td>
<td></td>
<td>• N = A UOW associated with this workload will be rejected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = A unit of work (UOW) associated with this workload can access the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>database.</td>
</tr>
<tr>
<td>MAXDEGREE</td>
<td>SMALLINT</td>
<td></td>
<td>Maximum degree of parallelism for the workload. The valid values are: 1 to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32767, and -1. If MAXIMUM DEGREE is DEFAULT, the value is -1.</td>
</tr>
<tr>
<td>SERVICECLASSNAME</td>
<td>VARCHAR (128)</td>
<td></td>
<td>Name of the service subclass to which a unit of work (associated with this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>workload) is assigned.</td>
</tr>
<tr>
<td>PARENTSERVICECLASSNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Name of the service superclass to which a unit of work (associated with this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>workload) is assigned.</td>
</tr>
<tr>
<td>COLLECTAGGACTDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what aggregate activity data should be captured for the workload</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base aggregate activity data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E = Collect extended aggregate activity data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td>COLLECTACTDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what activity data should be collected by the applicable event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = Activity data with details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S = Activity data with details and section environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• V = Activity data with details and values. Applies when the COLLECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>column is set to 'C'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• W = Activity data without details</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X = Activity data with details, section environment, and values</td>
</tr>
<tr>
<td>COLLECTACTPARTITION</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies where activity data is collected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C = Coordinator member of the activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = All members</td>
</tr>
<tr>
<td>COLLECTDEADLOCK</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies that deadlock events should be collect by the applicable event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H = Collect deadlock data with past activities only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• V = Collect deadlock data with past activities and values</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• W = Collect deadlock data without past activities and values</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Nullable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COLLECTLOCKTIMEOUT</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies that lock timeout events should be collect by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H = Collect lock timeout data with past activities only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Do not collect lock timeout data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• V = Collect lock timeout data with past activities and values</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• W = Collect lock timeout data without past activities and values</td>
</tr>
<tr>
<td>COLLECTLOCKWAIT</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies that lock wait events should be collect by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H = Collect lock wait data with past activities only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Do not collect lock wait data</td>
</tr>
<tr>
<td>LOCKWAITVALUE</td>
<td>INTEGER</td>
<td></td>
<td>Specifies the time in milliseconds a lock should wait before a lock event is collected by the applicable event monitor; 0 when COLLECTLOCKWAIT = 'N'</td>
</tr>
<tr>
<td>COLLECTACTMETRICS</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies the monitoring level for activities submitted by an occurrence of the workload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base activity metrics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E = Collect extended activity metrics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td>COLLECTUOWDATAOPTIONS</td>
<td>VARCHAR (32)</td>
<td></td>
<td>Specifies what unit of work data is collected by the applicable event monitor. The first position in the string represents whether the collection of unit of work data is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Enabled and collect base unit of work data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Not enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Starting from second position, each position in the string represents a specific extended option:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2 = Package Reference List</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3 = Executable ID list</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Each position representing an extended option is then set to one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = Extended option is included</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Extended option is not included</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Nullable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COLLECTUOWDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what unit of work data should be collected by the applicable event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base unit of work data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• P = Collect base unit of work data and the package list</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This column is deprecated. Information for the column is available from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>COLLECTUOWDATAOPTIONS.</td>
</tr>
<tr>
<td>EXTERNALNAME</td>
<td>VARCHAR (128)</td>
<td>Y</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>SECTIONACTUALSOPTIONS</td>
<td>VARCHAR (32)</td>
<td></td>
<td>Specifies what section actuals are collected during the execution of a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The first position in the string represents the whether the collection of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>section actuals is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Enabled and collect basic operator cardinality counts and statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>for each object referenced by the section (DML statements only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = Not enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The second position is always 'N' and reserved for for future use.</td>
</tr>
<tr>
<td>COLLECTAGGUOWDATA</td>
<td>CHAR (1)</td>
<td></td>
<td>Specifies what aggregate unit of work data should be captured for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>workload by the applicable event monitor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B = Collect base aggregate unit of work data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• N = None</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR (254)</td>
<td>Y</td>
<td>User-provided comments, or the null value.</td>
</tr>
</tbody>
</table>
Appendix A. General naming rules

Rules exist for the naming of all database objects, user names, passwords, groups, files, and paths. Some of these rules are specific to the platform you are working on.

For example, regarding the use of upper and lowercase letters in the names of objects that are visible in the file system (databases, instances, and so on):

- On UNIX platforms, names are case-sensitive. For example, /data1 is not the same directory as /DATA1 or /Data1.
- On Windows platforms, names are not case-sensitive. For example, \data1 is the same as \DATA1 and \Data1.

Unless otherwise specified, all names can include the following characters:

- The letters A through Z, and a through z, as defined in the basic (7-bit) ASCII character set. When used in identifiers for objects created with SQL statements, lowercase characters "a" through "z" are converted to uppercase unless they are delimited with quotation marks ("").
- 0 through 9.
- ! % ( ) { } . - ^ _ (underscore) @, #, $, and space.
- \ (backslash).

Restrictions

- Do not begin names with a number or with the underscore character.
- Do not use SQL reserved words to name tables, views, columns, indexes, or authorization IDs.
- Use only the letters defined in the basic ASCII character set for directory and file names. While your computer’s operating system might support different code pages, non-ASCII characters might not work reliably. Using non-ASCII characters can be a particular problem in distributed environment, where different computers might be using different code pages.
- There are other special characters that might work separately depending on your operating system and where you are working with the DB2 database. However, while they might work, there is no guarantee that they will work. It is not recommended that you use these other special characters when naming objects in your database.
- User and group names also must follow the rules imposed by specific operating systems. For example, on Linux and UNIX platforms, characters for user names and group names must be lowercase a through z, 0 through 9, and _ (underscore) for names not starting with 0 through 9.
- Lengths must be less than or equal to the lengths listed in “SQL and XML limits” in the SQL Reference.
- **Restrictions on the AUTHID identifier**: In DB2 Version 9.5 and later, you can have a 128-byte authorization ID. However, when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply. For example, the Linux and UNIX operating systems can contain up to 8 characters and the Windows operating systems can contain up to 30 characters for user IDs and group names. Therefore, while you can grant a 128-byte authorization ID, you cannot connect as a user that has that authorization ID. If you write your own security plug-in, you can use the
extended sizes for the authorization ID. For example, you can give your security plug-in a 30-byte user ID and it returns a 128-byte authorization ID during authentication that you can connect to.

You also must consider object naming rules, naming rules in an multicultural support environment, and naming rules in a Unicode environment.
Appendix B. Roles

Roles simplify the administration and management of privileges by offering an equivalent capability as groups but without the same restrictions.

A role is a database object that groups together one or more privileges and can be assigned to users, groups, PUBLIC, or other roles by using a GRANT statement, or can be assigned to a trusted context by using a CREATE TRUSTED CONTEXT or ALTER TRUSTED CONTEXT statement. A role can be specified for the SESSION_USER ROLE connection attribute in a workload definition.

Roles provide several advantages that make it easier to manage privileges in a database system:

- Security administrators can control access to their databases in a way that mirrors the structure of their organizations (they can create roles in the database that map directly to the job functions in their organizations).
- Users are granted membership in the roles that reflect their job responsibilities. As their job responsibilities change, their membership in roles can be easily granted and revoked.
- The assignment of privileges is simplified. Instead of granting the same set of privileges to each individual user in a particular job function, the administrator can grant this set of privileges to a role representing that job function and then grant that role to each user in that job function.
- A role’s privileges can be updated and all users who have been granted that role receive the update; the administrator does not need to update the privileges for every user on an individual basis.
- The privileges and authorities granted to roles are always used when you create views, triggers, materialized query tables (MQTs), static SQL and SQL routines, whereas privileges and authorities granted to groups (directly or indirectly) are not used.

This is because the DB2 database system cannot determine when membership in a group changes, as the group is managed by third-party software (for example, the operating system or an LDAP directory). Because roles are managed inside the database, the DB2 database system can determine when authorization changes and act accordingly. Roles granted to groups are not considered, due to the same reason groups are not considered.

- All the roles assigned to a user are enabled when that user establishes a connection, so all privileges and authorities granted to roles are taken into account when a user connects. Roles cannot be explicitly enabled or disabled.
- The security administrator can delegate management of a role to others.

All DB2 privileges and authorities that can be granted within a database can be granted to a role. For example, a role can be granted any of the following authorities and privileges:

- DBADM, SECADM, DATAACCESS, ACCESSCTRL, SQLADM, WLMADM, LOAD, and IMPLICIT_SCHEMA database authorities
- CONNECT, CREATETAB, CREATE_NOT_FENCED, BINDADD, CREATE_EXTERNAL_ROUTINE, or QUIESCE_CONNECT database authorities
- Any database object privilege (including CONTROL)
A user's roles are automatically enabled and considered for authorization when a user connects to a database; you do not need to activate a role by using the SET ROLE statement. For example, when you create a view, a materialized query table (MQT), a trigger, a package, or an SQL routine, the privileges that you gain through roles apply. However, privileges that you gain through roles granted to groups of which you are a member do not apply.

A role does not have an owner. The security administrator can use the WITH ADMIN OPTION clause of the GRANT statement to delegate management of the role to another user, so that the other user can control the role membership.

**Restrictions**

There are a few restrictions in the use of roles:

- A role cannot own database objects.
- Permissions and roles granted to groups are not considered when you create the following database objects:
  - Packages containing static SQL
  - Views
  - Materialized query tables (MQT)
  - Triggers
  - SQL Routines

Only roles granted to the user creating the object or to PUBLIC, directly or indirectly (such as through a role hierarchy), are considered when creating these objects.
Appendix C. Trusted contexts and trusted connections

A trusted context is a database object that defines a trust relationship for a connection between the database and an external entity such as an application server.

The trust relationship is based upon the following set of attributes:

- System authorization ID: Represents the user that establishes a database connection
- IP address (or domain name): Represents the host from which a database connection is established
- Data stream encryption: Represents the encryption setting (if any) for the data communication between the database server and the database client

When a user establishes a database connection, the DB2 database system checks whether the connection matches the definition of a trusted context object in the database. When a match occurs, the database connection is said to be trusted.

A trusted connection cannot be established if the connection is to a local database using inter-process communication (IPC).

A trusted connection allows the initiator of this trusted connection to acquire additional capabilities that may not be available outside the scope of the trusted connection. The additional capabilities vary depending on whether the trusted connection is explicit or implicit.

The initiator of an explicit trusted connection has the ability to:

- Switch the current user ID on the connection to a different user ID with or without authentication
- Acquire additional privileges via the role inheritance feature of trusted contexts

An implicit trusted connection is a trusted connection that is not explicitly requested; the implicit trusted connection results from a normal connection request rather than an explicit trusted connection request. No application code changes are needed to obtain an implicit connection. Also, whether you obtain an implicit trusted connection or not has no effect on the connect return code (when you request an explicit trusted connection, the connect return code indicates whether the request succeeds or not). The initiator of an implicit trusted connection can only acquire additional privileges via the role inheritance feature of trusted contexts; they cannot switch the user ID.

How using trusted contexts enhances security

The three-tiered application model extends the standard two-tiered client and server model by placing a middle tier between the client application and the database server. It has gained great popularity in recent years particularly with the emergence of web-based technologies and the Java™ 2 Enterprise Edition (J2EE) platform. An example of a software product that supports the three-tier application model is IBM WebSphere Application Server (WAS).

In a three-tiered application model, the middle tier is responsible for authenticating the users running the client applications and for managing the interactions with the database server. Traditionally, all the interactions with the database server
occur through a database connection established by the middle tier using a combination of a user ID and a credential that identify that middle tier to the database server. This means that the database server uses the database privileges associated with the middle tier's user ID for all authorization checking and auditing that must occur for any database access, including access performed by the middle tier on behalf of a user.

While the three-tiered application model has many benefits, having all interactions with the database server (for example, a user request) occur under the middle tier's authorization ID raises several security concerns, which can be summarized as follows:

- **Loss of user identity**
  Some enterprises prefer to know the identity of the actual user accessing the database for access control purposes.

- **Diminished user accountability**
  Accountability through auditing is a basic principle in database security. Not knowing the user's identity makes it difficult to distinguish the transactions performed by the middle tier for its own purpose from those performed by the middle tier on behalf of a user.

- **Over granting of privileges to the middle tier's authorization ID**
  The middle tier's authorization ID must have all the privileges necessary to execute all the requests from all the users. This has the security issue of enabling users who do not need access to certain information to obtain access anyway.

- **Weakened security**
  In addition to the privilege issue raised in the previous point, the current approach requires that the authorization ID used by the middle tier to connect must be granted privileges on all resources that might be accessed by user requests. If that middle-tier authorization ID is ever compromised, then all those resources will be exposed.

- **“Spill over” between users of the same connection**
  Changes by a previous user can affect the current user.

Clearly, there is a need for a mechanism whereby the actual user's identity and database privileges are used for database requests performed by the middle tier on behalf of that user. The most straightforward approach of achieving this goal would be for the middle-tier to establish a new connection using the user's ID and password, and then direct the user's requests through that connection. Although simple, this approach suffers from several drawbacks which include the following:

- **Inapplicability for certain middle tiers.** Many middle-tier servers do not have the user authentication credentials needed to establish a connection.

- **Performance overhead.** There is an obvious performance overhead associated with creating a new physical connection and re-authenticating the user at the database server.

- **Maintenance overhead.** In situations where you are not using a centralized security set up or are not using single sign-on, there is maintenance overhead in having two user definitions (one on the middle tier and one at the server). This requires changing passwords at different places.

The trusted contexts capability addresses this problem. The security administrator can create a trusted context object in the database that defines a trust relationship between the database and the middle-tier. The middle-tier can then establish an explicit trusted connection to the database, which gives the middle tier the ability to switch the current user ID on the connection to a different user ID, with or without authentication. In addition to solving the end-user identity assertion
trusted contexts offer another advantage. This is the ability to control
when a privilege is made available to a database user. The lack of control on when
privileges are available to a user can weaken overall security. For example,
privileges may be used for purposes other than they were originally intended. The
security administrator can assign one or more privileges to a role and assign that
role to a trusted context object. Only trusted database connections (explicit or
implicit) that match the definition of that trusted context can take advantage of the
privileges associated with that role.

Enhancing performance

When you use trusted connections, you can maximize performance because of the
following advantages:

• No new connection is established when the current user ID of the connection is
  switched.

• If the trusted context definition does not require authentication of the user ID to
  switch to, then the overhead associated with authenticating a new user at the
database server is not incurred.

Example of creating a trusted context

Suppose that the security administrator creates the following trusted context object:

```
CREATE TRUSTED CONTEXT CTX1
  BASED UPON CONNECTION USING SYSTEM AUTHID USER2
  ATTRIBUTES (ADDRESS '192.0.2.1')
  DEFAULT ROLE managerRole
  ENABLE
```

If user user1 requests a trusted connection from IP address 192.0.2.1, the DB2
database system returns a warning (SQLSTATE 01679, SQLCODE +20360) to
indicate that a trusted connection could not be established, and that user user1
simply got a non-trusted connection. However, if user user2 requests a trusted
connection from IP address 192.0.2.1, the request is honored because the connection
attributes are satisfied by the trusted context CTX1. Now that use user2 has
established a trusted connection, he or she can now acquire all the privileges and
authorities associated with the trusted context role managerRole. These privileges
and authorities may not be available to user user2 outside the scope of this trusted
connection.
Appendix D. DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

• Online DB2 documentation in IBM Knowledge Center:
  – Topics (task, concept, and reference topics)
  – Sample programs
  – Tutorials
• Locally installed DB2 Information Center:
  – Topics (task, concept, and reference topics)
  – Sample programs
  – Tutorials
• DB2 books:
  – PDF files (downloadable)
  – PDF files (from the DB2 PDF DVD)
  – Printed books
• Command-line help:
  – Command help
  – Message help

Important: The documentation in IBM Knowledge Center and the DB2 Information Center is updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 documentation in IBM Knowledge Center.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

The DB2 Information Development team values your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 Information Development team reads all of your feedback but cannot respond to you directly. Provide specific examples wherever possible to better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use the db2docs@ca.ibm.com email address to contact DB2 Customer Support. If you have a DB2 technical issue that you cannot resolve by using the documentation, contact your local IBM service center for assistance.
**DB2 technical library in hardcopy or PDF format**

You can download the DB2 technical library in PDF format or you can order in hardcopy from the IBM Publications Center.

English and translated DB2 Version 10.5 manuals in PDF format can be downloaded from DB2 database product documentation at www.ibm.com/support/docview.wss?rs=71&uid=swg27009474.

The following tables describe the DB2 library available from the IBM Publications Center at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss. Although the tables identify books that are available in print, the books might not be available in your country or region.

The form number increases each time that a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed in the following tables.

The DB2 documentation online in IBM Knowledge Center is updated more frequently than either the PDF or the hardcopy books.

Table 291. DB2 technical information

<table>
<thead>
<tr>
<th>Name</th>
<th>Form number</th>
<th>Available in print</th>
<th>Availability date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrative API Reference</td>
<td>SC27-5506-00</td>
<td>Yes</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Administrative Routines and Views</td>
<td>SC27-5507-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Call Level Interface Guide and Reference</td>
<td>SC27-5511-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Call Level Interface Guide and Reference</td>
<td>SC27-5512-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Call Level Interface Guide and Reference</td>
<td>SC27-5511-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Call Level Interface Guide and Reference</td>
<td>SC27-5512-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Command Reference</td>
<td>SC27-5508-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Database Administration Concepts and</td>
<td>SC27-4546-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Configuration Reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Movement Utilities Guide and Reference</td>
<td>SC27-5528-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Database Monitoring Guide and Reference</td>
<td>SC27-4547-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Data Recovery and High Availability Guide</td>
<td>SC27-5529-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>and Reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 Workload Management Guide and Reference</td>
<td>SC27-5520-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Developing ADO.NET and OLE DB Applications</td>
<td>SC27-4549-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Developing Embedded SQL Applications</td>
<td>SC27-4550-00</td>
<td>Yes</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Name</td>
<td>Form number</td>
<td>Available in print</td>
<td>Availability date</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------------</td>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Developing Java Applications</td>
<td>SC27-5503-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Developing Perl, PHP, Python, and Ruby on Rails Applications</td>
<td>SC27-5504-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Developing RDF Applications for IBM Data Servers</td>
<td>SC27-5505-00</td>
<td>Yes</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Developing User-defined Routines (SQL and External)</td>
<td>SC27-5501-00</td>
<td>Yes</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Getting Started with Database Application Development</td>
<td>GI13-2084-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Getting Started with DB2 Installation and Administration on Linux and Windows</td>
<td>GI13-2085-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Globalization Guide</td>
<td>SC27-5531-00</td>
<td>No</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Installing DB2 Servers</td>
<td>GC27-5514-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Installing IBM Data Server Clients</td>
<td>GC27-5515-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Message Reference Volume 1</td>
<td>SC27-5523-00</td>
<td>No</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Message Reference Volume 2</td>
<td>SC27-5524-00</td>
<td>No</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Net Search Extender Administration and User’s Guide</td>
<td>SC27-5526-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Partitioning and Clustering Guide</td>
<td>SC27-5532-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>pureXML Guide</td>
<td>SC27-5521-00</td>
<td>No</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Spatial Extender User’s Guide and Reference</td>
<td>SC27-5525-00</td>
<td>No</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>SQL Procedural Languages: Application Enablement and Support</td>
<td>SC27-5502-00</td>
<td>No</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>SQL Reference Volume 1</td>
<td>SC27-5509-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>SQL Reference Volume 2</td>
<td>SC27-5510-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Text Search Guide</td>
<td>SC27-5527-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Troubleshooting and Tuning Database Performance</td>
<td>SC27-4548-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Upgrading to DB2 Version 10.5</td>
<td>SC27-5513-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>What’s New for DB2 Version 10.5</td>
<td>SC27-5519-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>XQuery Reference</td>
<td>SC27-5522-01</td>
<td>No</td>
<td>1 October 2014</td>
</tr>
<tr>
<td>Name</td>
<td>Form number</td>
<td>Available in print</td>
<td>Availability date</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------</td>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Installing and Configuring DB2</td>
<td>SC27-5517-00</td>
<td>Yes</td>
<td>28 July 2013</td>
</tr>
<tr>
<td>Connect Servers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 Connect User’s Guide</td>
<td>SC27-5518-01</td>
<td>Yes</td>
<td>1 October 2014</td>
</tr>
</tbody>
</table>

### Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

**Procedure**

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where `sqlstate` represents a valid five-digit SQL state and `class code` represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

### Accessing DB2 documentation online for different DB2 versions

You can access online the documentation for all the versions of DB2 products in IBM Knowledge Center.

**About this task**

All the DB2 documentation by version is available in IBM Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSEPGG/welcome. However, you can access a specific version by using the associated URL for that version.

**Procedure**

To access online the DB2 documentation for a specific DB2 version:

- To access the DB2 Version 10.5 documentation, follow this URL: http://www.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.kc.doc/welcome.html.
Terms and conditions

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the previous instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

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

This information was developed for products and services offered in the U.S.A.
Information about non-IBM products is based on information available at the time
of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in
other countries. Consult your local IBM representative for information about the
products and services currently available in your area. Any reference to an IBM
product, program, or service is not intended to state or imply that only that IBM
product, program, or service may be used. Any functionally equivalent product,
program, or service that does not infringe any IBM intellectual property right may
be used instead. However, it is the user’s responsibility to evaluate and verify the
operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter
described in this document. The furnishing of this document does not grant you
any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information,
contact the IBM Intellectual Property Department in your country or send
inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other
country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS
PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER
EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS
FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or
implied warranties in certain transactions; therefore, this statement may not apply
to you.

This information could include technical inaccuracies or typographical errors.
Changes are periodically made to the information herein; these changes will be
incorporated in new editions of the publication. IBM may make improvements,
changes, or both in the product(s) and/or the program(s) described in this
publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for
convenience only and do not in any manner serve as an endorsement of those
websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

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

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

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

**Trademarks**

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

The following terms are trademarks or registered trademarks of other companies

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

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

A

activities
  aggregating metrics across children 349
  analysis example 61
  application of work actions 108
  assignment to work classes 59
  business intelligence reports
    controlling (scenario) 153
  canceling
    procedure 222, 279
    scenarios 352, 355, 358
  controlling
    scenario 154
    scenario (business intelligence reports) 153
data collection
  example 281
  procedure 277
  Design Advisor 279
  long-running
    scenario 352
dlows estimated cost and long run time
    scenario 357
  mapping to service classes 72
  monitor elements
    act_throughput 440
    act_total 441
    activity_collected 442
    activity_id 442
    activity_secondary_id 443
    activity_type 443
    coord_act_aborted_total 456
    coord_act_completed_total 456
    coord_act_rejected_total 463
    parent_activity_id 471
  overview 15
  queuing 138
  rogue 280
  states in service classes 77
  thresholds 128
activities monitor elements (continued)
  CONCURRENTDBCOORDACTIVITIES threshold (continued)
    concurrentdbcoordactivities_wl_was_threshold_queued 454
    concurrentdbcoordactivities_wl_was_threshold_value 454
    concurrentdbcoordactivities_wl_was_threshold_violated 455
    concurrentdbcoordactivities_db_threshold_id 448
    concurrentdbcoordactivities_subclass_threshold_value 450
    concurrentdbcoordactivities_subclass_threshold_queued 450
    concurrentdbcoordactivities_subclass_threshold_violated 451
    concurrentdbcoordactivities_superclass_threshold_id 451
    concurrentdbcoordactivities_superclass_threshold_queued 452
    concurrentdbcoordactivities_superclass_threshold_violated 452
    concurrentdbcoordactivities_superclass_threshold_violated 453
    coord_act_est_cost_avg 457
    coord_act_exec_time_avg 458
    coord_act_interarrival_time_avg 459
    coord_act_lifetime_avg 460
    coord_act_queue_time_avg 462
    destination_service_class_id 468
    histograms
      histogram_type 468
      number_in bin 471
      top 493
  identifiers
    arm_correlator 445
    bin_id 445
    db_work_action_set_id 467
    db_work_class_id 467
    sc_work_action_set_id 479
    sc_work_class_id 480
    service_class_id 481
    work_action_set_id 508
    work_class_id 509
  locks
    uow_lock_wait_time 499
  log space
    uow_log_space_used 500
  names
    service_subclass_name 482
    service_superclass_name 483
    work_action_set_name 509
    work_class_name 510
  num_remaps 470
  partitions
    coord_partition_num 463
  queries
    queue_assignments_total 473
    queue_size_top 473
    queue_time_total 473

© Copyright IBM Corp. 2007, 2014
activities monitor elements (continued)

ranges
- bottom 445
- request_exec_time_avg 474
routines
- routine_id 475
rows
- rows_fetched 475
- rows_modified 476
- rows_returned 477
sections
- section_env 481
- source_service_class_id 484
statements
- stmt_invocation_id 485
thresholds
- num_threshold_violations 470
- thres_violations 487
- threshold_action 488
- threshold_domain 488
- threshold_maxvalue 489
- threshold_name 490
- threshold_predicate 490
- threshold_queue_size 491
- thresholdid 491
time
- prep_time 472
- time_completed 492
- time_created 492
- time_ofViolation 492
- time_started 493
time stamps
- activate_timestamp 441
- statistics_timestamp 485
units of work (UOW)
- parent_uow_id 471
- uow_comp_status 496
- uow_completed_total 495
- uow_elapsed_time 497
- uow_id 497
- uow_lifetime_avg 498
- uow_start_time 500
- uow_status 501
- uow_stop_time 502
- uow_throughput 502
watermarks
- act_cpu_time_top 438
- act_rows_read_top 439
- concurrent_act_top 445
- concurrent_connection_top 446
- concurrent_wlo_act_top 447
- concurrent_wlo_top 447
- coord_act_lifetime_top 461
- cost_estimate_top 464
- rows_returned_top 479
- temp_tablespace_top 486
- uow_total_time_top 503
WLM dispatcher
- cpu_limit 464
- cpu_share_type 465
- cpu_shares 465
- cpu_utilization 465
- cpu_velocity 466
- estimated_cpu_entitlement 468
- total_disp_run_queue_time 493
workload management
- overview 438

workload management (continued)

wl_work_action_set_id 504
wl_work_class_id 505
wlm_queue_assignments 505
wlm_queue_time_total 507
workloads
- wlo_completed_total 508
- workload_id 510
- workload_name 511
- workload_occurrence_id 512
- workload_occurrence_state 513
activity event monitors
- WLM 237
activity throughput
- monitor elements
  - act_throughput 440
ACTIVITYTOTALTIME activity threshold
details 129
agents
- service class usage 91
aggregate thresholds
- overview 137
aggregation
data
- example 235
AGGSQLTEMPSPACE threshold 138
AIX Workload Manager
- integrating DB2 workload management 285
APIs
- sqlesi
  - workload assignment 38
AUTHID identifier
  - restrictions 533

B
- bins
  - example 265
buffer pool priority
  - service classes 76

C
CALL statement
  - classification by schema 51
catalog views
  - HISTOGRAMTEMPLATEBINS 519
  - HISTOGRAMTEMPLATES 519
  - HISTOGRAMTEMPLATEUSE 520
  - SERVICECLASSES 521
  - THRESHOLDS 523
  - WORKACTIONS 525
  - WORKACTIONSETS 527
  - WORKCLASSES 528
  - WORKCLASSSETS 528
  - WORKLOADAUTH 529
  - WORKLOADCONNATTR 529
  - WORKLOADS 529
column-organized tables
  - query concurrency 213
commands
  - SET WORKLOAD
    - details 514
    - taking corrective actions 28
concurrency management 213
CONCURRENTDBCOORDACTIVITIES aggregate threshold 139
CONCURRENTWORKLOADACTIVITIES aggregate threshold 140
CONCURRENTWORKLOADOCCURRENCES aggregate threshold 142
configuration parameters
   wlm_collect_int 515
   wlm_disp_concur 517
   wlm_disp_cpu_shares 517
   wlm_disp_min_util 518
   wlm_dispatcher 516
CONNECTIONIDLETIME connection threshold 128
connections
   assigning
   default administration workload 28
   workload connection attributes with multiple values 47
   workloads 23
   mapping to workloads 42
   states in service classes 77
   transient 144
CPU limit
   monitor elements
   cpu_limit 464
   setting 204
CPU share type
   monitor elements
   cpu_share_type 465
CPU shares
   monitor elements
   cpu_shares 465
CPU utilization
   monitor elements
   cpu_utilization 465
CPU velocity
   monitor elements
   cpu_velocity 466
CPUTIME activity threshold 130
CPUTIMEINSC activity threshold 131

DB2 workload management (continued)
activities (continued)
   discovering 332
   identifying activities with low estimated costs and high run times (example) 357
   identifying long-running activities (example) 352
   importing information into Design Advisor 279
   isolating 302
   mapping to service classes 72
   overview 15
   remapping 149
   rogue 280
   thresholds 128
   activity queues 138
agents
   usage by service class (example) 91
   aggregating metrics activities 349
AIX Workload Manager integration 285
canceling activities 355
concurrency control 217
connections
   assignment to workloads 23
   states in service classes 77
CPU limit
   details 190
   setting 204
CPU shares 188
DB2 Governor relationship 4
DDL statements 18
default objects for concurrency control 217
dispatch concurrency level
details 208
setting 209
dispatcher
details 159
enabling 177
ensuring maximum scheduling accuracy 178
multimember environments 175
overview 158
domains 1
event monitors
   overview 237
extended aggregates 340
frequently asked questions 4
hard CPU shares
details 178
histogram templates
   altering 264
   creating 263
   dropping 264
histograms
   computing averages and standard deviation (example) 265
   creating 319
   overview 258
historical analysis tool 266
historical data 337
integration with operating system workload managers 285
Linux workload management integration 290
management stage 67, 91, 117
metrics 270
minimum CPU utilization
details 205
monitoring
application delays 328
data collection 297
event monitors 237
overview 225
real-time 225
service class tiers 256
system behavior at different levels (example) 231
work 225
object ownership 18
overview 1
performance modeling 281
priority aging
details 145
sample scripts 149
Query Patroller 4
reports 337
sample application 347
service class tiering
overview 145
sample scripts 149
service classes
altering 81
analyzing system slowdown (example) 89
creating 79
dropping 83
entities not tracked 78
example 84
obtaining point-in-time statistics (example) 234
prefetch priority 76
resource assignment 67
service subclasses 67
SET WORKLOAD command 514
soft CPU shares
details 185
statistics
collecting using statistics event monitor 267
overview 246
resetting 269
stored procedures
WLM_CANCEL_ACTIVITY 245
WLM_CAPTURE_ACTIVITY_IN_PROGRESS 245
WLM_COLLECT_STATS 245
WLM_SET_CLIENT_INFO 245
table functions
aggregating data (example) 235
obtaining WLM threshold queue information (example) 236
operational information 225
understanding what is running on data server (example) 230
using with snapshot monitor table functions 271
thresholds
activity 128
ACTIVITYTOTALTIME 129
aggregate 137
AGGSQLTEMPSPACE 138
altering 124
CONCURRENTDBCOORDACTIVITIES 139
thresholds (continued)
CONCURRENTWORKLOADACTIVITIES 140
CONCURRENTWORKLOADOCCURRENCES 142
connection 128
CONNECTIONIDLETIME 128
controlling work 117
CPUTIME 130
CPUTIMEINSC 131
creating 123
DATATAGINSC 132
dropping 125
ESTIMATEDSQLCOST 133
evaluation order 121
managing database resources across departments (example) 126
overview 117
remapping activities 145, 155
rogue activities 309
scope resolution 129
SQLROWSREAD 134
SQLROWSREADINSC 135
SQLROWSRETURNED 136
SQLTEMPSPACE 137
summary 120
TOTALMEMBERCONNECTIONS 143
TOTALMEMBERCONNECTIONS 144
UOWTOTALTIME 145
violation monitoring 272
work action set and database threshold (example) 114
tuning
capacity planning data available (example) 360
capacity planning data unavailable (example) 362
tutorial 297
units of work
workload assignment when multiple workloads exist (example) 44
WORKlicate privilege on workloads
granting 37
revoking 38
WLMADM authority
overview 3
work
controlling 67
identifying 15
work action sets
altering 102
creating 101
determining types of work being run (example) 116
disabling 103
dropping 104
overview 94
using with database threshold (example) 114
work actions
altering 99
assigning to database activities 108
creating 96
defining for work action set 104
disabling 100
dropping 101
thresholds 108
work class sets
altering 58
creating 57
DML activity management (example) 62
dropping 58
DB2 workload management (continued)
work class sets (continued)
overview 51
work classes
altering 56
classification of CALL statement by schema 51
creating 53
defined with ALL keyword (example) 63
dropping 57
evaluation order 58
overview 48
workload management dispatcher
CPU limit 190
hard CPU shares 178
minimum CPU utilization 205
soft CPU shares 185
workloads
altering 32
analyzing system slowdown (example) 348
assignment (example) 38
assignment when multiple workloads exist (example) 44
assignment when workload attributes have multiple values (example) 47
connection assignment to default administration workload 28
creating 30
database access permission 35
database access prevention 36
default 25
disabling 34
dropping 34
enabling 33
overview 19
DDL
statements
DB2 workload management 18
default maintenance service superclass overview 70
default objects for concurrency control 217
default service subclasses 70
default system service superclass 70
default user service superclass 70
default workloads 25
Design Advisor
importing activity information 279
dispatch concurrency level setting 209
documentation
PDF files 542
printed 542
terms and conditions of use 545

E
estimated CPU entitlement
monitor elements
estimated_cpu_entitlement 468
ESTIMATEDSQLCOST activity threshold 133
evaluation order
DB2 workload management thresholds 121
workloads 23
event monitors
activities
collecting data 277
DB2 workload management statistics collection 267

event monitors (continued)
threshold violations
monitoring 272
types 237
examples
mapping connections to workloads 42
service classes 84
work action set and threshold 114
work class defined with ALL keyword 63
work class set management of activities 62

F
file names
general 533
functions
table
MON_GET_ACTIVITY 416
MON_GET_ACTIVITY_DETAILS 377
MON_GET_AGENT 396
MON_GET_QUEUE_STATS 393
MON_GET_SERVICE_SUBCLASS_STATS 406
MON_GET_SERVICE_SUPERCLASS_STATS 413
MON_GET_WORK_ACTION_SET_STATS 414
MON_GET_WORKLOAD_STATS 432
WLM_GET_ACTIVITY_DETAILS 372
WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES 403

H
hard CPU shares
enabling 188
setting 188
help
SQL statements 544
histogram templates
altering 264, 269
creating 263
dropping 264
histograms
element 265
monitor elements
histogram_type 468
number_in_bin 471
top 493
overview 258

I
IBM Knowledge Center
DB2 documentation versions 544
identifiers
monitor elements
arm_correlator 445
bin_id 445
db_work_action_set_id 467
db_work_class_id 467
sc_work_action_set_id 479
sc_work_class_id 480
service_class_id 481
work_action_set_id 508
work_class_id 509
in-service-class thresholds
priority aging 145
Linux
workload management integration with DB2 workload management 290
locks
monitor elements
  uow_lock_wait_time 499
logs
monitor elements
  uow_log_space_used 500

metrics
  DB2 workload management objects 270
minimum CPU utilization
  setting 207
MON_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure 368
MON_COLLECT_STATS stored procedure 370
MON_GET_ACTIVITY table function 416
MON_GET_ACTIVITY_DETAILS table function 377
MON_GET_AGENT table function 396
MON_GET_QUEUE_STATS table function 393
MON_GET_SERVICE_SUBCLASS_STATS table function 406
MON_GET_SERVICE_SUPERCLASS_STATS table function
  function 413
MON_GET_WORK_ACTION_SET_STATS table function 414
MON_GET_WORKLOAD_STATS table function 432
monitor elements
  concurrentdbcoordactivities_db_threshold_value 449
  concurrentdbcoordactivities_db_threshold_violated 449
  concurrentdbcoordactivities_subclass_threshold_id 449
monitoring
data
  workload management 243
  entities not tracked by service class 78
  historical trends 237
overview 225
priority aging 256
real-time 225
service class tiers 256
workload management dispatcher
details 210
performance 210
workload types 210

names
monitor elements
  service_subclass_name 482
  service_superclass_name 483
  work_action_set_name 509
  work_class_name 510
naming conventions
general 533
notices 547
notifications
  threshold violations 274

online DB2 documentation
  IBM Knowledge Center 544

operating systems
  integrating DB2 workload management 285
ownership
  DB2 workload management objects 18

partitioned database environments
monitor elements
  coord_partition_num 463
paths
  naming rules 533
performance
  DB2 workload management
    examples 360, 362
    performance modeling 281
prefetching
  priority of service classes 76
privileges
  roles 535
procedures
  MON_CAPTURE_ACTIVITY_IN_PROGRESS 368
  MON_COLLECT_STATS 370
  WLM_CANCEL_ACTIVITY 367
  WLM_SET_CLIENT_INFO 435
queries
monitor elements
  queue_assignments_total 473
  queue_size_top 473
  queue_time_total 473
query concurrency
default management 213
queues
  prefetch 76
ranges
monitor elements
  bottom 445
REMAP ACTIVITY action
defining 145
sample scripts 149
remapping activities
details 155
sample scripts 149
roles
details 535
routine_id monitor element 475
routines
monitor elements
  routine_id 475
  WLM_CANCEL_ACTIVITY example 352
rows
monitor elements
  rows_fetched 475
  rows_modified 476
  rows_returned 477
  rows_returned_top 479
scenarios
  activities
    aggregating metrics across children 349
    cancelling 355
    aggregating activity metrics across children 349
    cancelling activities 355
schemes
  classification of CALL statements 51
sections
  monitor elements
    section_env 481
security
  trusted contexts 537
service classes
  activity states 77
  altering
    changes occur at statistics reset 269
  procedure 81
  analyzing system slowdown 89
  buffer pool priority 76
  connection states 77
  creating 79
  default service subclasses 70
  default service superclasses 70
  dropping 83
  entities not tracked by 78
  examples 84, 89, 234
  mapping activities 72
  point-in-time statistics 234
  prefetch priority 76
service subclasses
  altering 81
  creating 79
  default 217
  dropping 83
  monitoring data 243
service superclasses
  altering 81
  creating 79
  dropping 83
  monitoring data 243
SET WORKLOAD command
  assigning connection to default administration
  workload 28
  details 514
snapshot monitoring
  supplementing table functions 271
soft CPU shares
  enabling 188
  setting 188
SQL statements
  help
    displaying 544
    monitor elements
      stmt_invocation_id 485
sqlseti API
  workload assignment example 38
SQLROWSREAD activity threshold 134
SQLROWSREADINSC activity threshold 135
SQLROWSRETURNED activity threshold 136
SQLTEMPSPACE activity threshold
  details 137
statement invocation identifier monitor element 485
statistics
  collection
    workload management 267
  (continued)
    DB2 workload management objects 246
    event monitor 237
stored procedures
  MON_CAPTURE_ACTIVITY_IN_PROGRESS 368
  MON_COLLECT_STATS 370
  WLM_CANCEL_ACTIVITY 245
  WLM_CAPTURE_ACTIVITY_IN_PROGRESS 245
  WLM_COLLECT_STATS 245
  WLM_SET_CLIENT_INFO 245
SYSDEFAULTMAINTENANCECLASS service superclass 70
SYSDEFAULTSYSTEMCLASS service superclass 70
SYSDEFAULTUSERCLASS service superclass 70
T
  table functions
    MON_GET_ACTIVITY 416
    MON_GET_AGENT 396
    MON_GET_QUEUE_STATS 393
    MON_GET_SERVICE_SUBCLASS_STATS 406
    MON_GET_SERVICE_SUPERCLASS_STATS 413
    MON_GET_WORK_ACTION_SET_STATS 414
    MON_GET_WORKLOAD_STATS 432
  monitoring
    different levels 231
    snapshot monitor 271
    WLM_COLLECT_STATS 269
    WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES 403
workload management
  aggregating data 235
  determining what is running 230
  threshold queue information 236
table spaces
  SQLTEMPSPACE threshold 137
terms and conditions
  publications 545
threshold violations event monitor
  example 237
thresholds
  action 117
  activity 128
  activity queuing 138
  ACTIVITYTOTALTIME 129
  aggregate 117, 137
  AGGSQLTEMPSPACE 138
  altering 124
  CONCURRENTDBCOORDACTIVITIES 139
  CONCURRENTWORKLOADACTIVITIES 140
  CONCURRENTWORKLOADOCCURRENCES 142
  connection 128
  CONNECTIONIDLETIME 128
  CPUTIME 130
  CPUTIMEINSC 131
  creating 123
  DATATAGINSC 132
  default 217
  domain 120
  dropping 125
  enforcement scope 120
  ESTIMATEDSQLCOST 133
  evaluation order 121
examples
  usage 126
  work action set and database threshold 114
thresholds (continued)
monitor elements
  num_threshold_violations 470
  thresh_violations 487
  threshold_action 488
  threshold_domain 488
  threshold_maxvalue 489
  threshold_name 490
  threshold_predicate 490
  threshold_queue_size 491
  thresholdid 491
monitoring violations 272
overview 117
remapping activities 145, 155
scope resolution of activity 129
SQLROWSREAD 134
SQLROWSREADINC 135
SQLROWSRETURNED 136
SQLTEMPSPACE 137
TOTALMEMBERCONNECTIONS 143
TOTALSCMEMBERCONNECTIONS 144
unit of work 145
UOWTOTALTIME 145
violations
  email notifications 274
monitoring 272
work actions 108
work classifications 60
time
monitor elements
  prep_time 472
  time_completed 492
  time_created 492
  time_ofViolation 492
  time_started 493
time stamps
monitor elements
  activate_timestamp 441
  statistics_timestamp 485
  uow_start_time 500
  uow_stop_time 502
total completed units of work
monitor elements
  uow_completed_total 495
total dispatcher queue time
monitor elements
  total_disp_run_queue_time 493
TOTALMEMBERCONNECTIONS connection threshold 143
TOTALSCMEMBERCONNECTIONS connection threshold 144
trusted connections
overview 537
trusted contexts
overview 537

U
units of work
  assignment to default workload 25
  matching to workload 44
monitor elements
  parent_uow_id 471
  uow_comp_status 496
  uow_completed_total 495
  uow_elapsed_time 497
  uow_id 497
  uow_lifetime_avg 498
units of work (continued)
monitor elements (continued)
  uow_lock_wait_time 499
  uow_log_space_used 500
  uow_start_time 500
  uow_status 501
  uow_stop_time 502
  uow_throughput 502
thresholds 145
units of work lifetime average
monitor elements
  uow_lifetime_avg 498
units of work throughput
monitor elements
  uow_throughput 502
UOWTOTALTIME threshold 145

W
watermark monitor elements
  act_cpu_time_top 438
  act_rows_read_top 439
  concurrent_act_top 445
  concurrent_connection_top 446
  concurrent_wlo_act_top 447
  concurrent_wlo_top 447
  coord_act_lifetime_top 461
  cost_estimate_top 464
  rows_returned_top 479
  temp_tablespace_top 486
  uow_total_time_top 503
WLM_CANCEL_ACTIVITY procedure 367
WLM_COLLECT_STATS procedure
  resetting statistics 269
wlm_ctx_conc configuration parameter
details 517
setting 209
wlm_ctx_cpu_shares configuration parameter 517
wlm_ctx_min_util configuration parameter
details 518
setting 207
wlm_dispatcher configuration parameter
details 516
setting 177
WLM_GET_ACTIVITY_DETAILS table function
aggregating data (example) 235
details 403
WLM_GET_ACTIVITY_DETAILS table function
examples
  investigating agent usage by service class 91
  WLM queue information 226
WLM_GET_SERVICE_CLASS_AGENTS table function
WLM_GET_SERVICE_CLASS_AGENTS table function
  examples
  analyzing system slowdown 89, 348
  obtaining point-in-time statistics 234
WLM_GET_WORK_ACTION_SET_STATS table function
analyzing workloads (examples) 61
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function
WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function
  examples
  aggregating data 235
  identifying long-running activities 352
WLM_SET_CLIENT_INFO procedure 435
WLMADM (workload administration) authority
   details 3
work action sets
   altering 102
   concurrency control 110
   creating 101
   default 217
   disabling 103
   domain and permitted work actions 104
   dropping 104
   examples
      association with other objects 92
      determining types of work being run 116
   work action set and database threshold 114
   overview 94
   work actions specifying thresholds 108
   workload level
      concurrency control 110
work actions
   altering 99
   assigning to database activities 108
   association with other objects (example) 92
   creating 96
   default 217
   disabling 100
   dropping 101
   thresholds 108
   work action sets 104
work class sets
   altering 58
   association with other objects (example) 92
   creating 57
   default 217
   dropping 58
   managing DML activities (example) 62
   overview 51
   work class evaluation order 58
work class evaluation order
   58
work classes
   altering 56
   assigning activities 59
   creating 53
   default 217
   dropping 57
   evaluation order 58
   examples
      association with other objects 92
      defined with ALL keyword 63
   overview 48
   thresholds 60
workload management
   examples
      cancellation of all activities 358
      disconnecting all applications 359
      threshold violation email notifications 274
workload management dispatcher (continued)
   ensuring maximum scheduling accuracy 178
   hard CPU shares 178
   micropartition environments 175
   minimum CPU utilization
      details 205
      setting 207
monitor elements
   cpu_limit 464
   cpu_share_type 465
   cpu_shares 465
   cpu_utilization 465
   cpu_velocity 466
   estimated_cpu_entitlement 468
   total_disp_run_queue_time 493
monitoring 210
multimember environments 175
overview 158
soft CPU shares 185
workload management dispatcher configuration parameter 516
workload manager dispatcher CPU shares configuration parameter 517
workload manager dispatcher minimum CPU utilization configuration parameter 518
workload manager dispatcher thread concurrency configuration parameter 517
workloads
   altering 32
   assignment
      details 23
      examples 38
   connection assignment to the default administration workload 28
   creating 30
   database access
      permitting 35
      preventing 36
   default 25
   disabling 34
   dropping 34
   enabling 33
   evaluation order 23
   examples
      analyzing system slowdown 348
      assignment when multiple workloads exist 44
      assignment when workload attributes have multiple values 47
      assignment when workload attributes have single values 42
monitor elements
   wlo_completed_total 508
   workload_id 510
   workload_name 511
   workload_occurrence_id 512
   workload_occurrence_state 513
monitoring data 243
overview 19
position in workload list 23
preventing database access 36
USAGE privilege
   granting 37
   revoking 38
work action set comparison 112