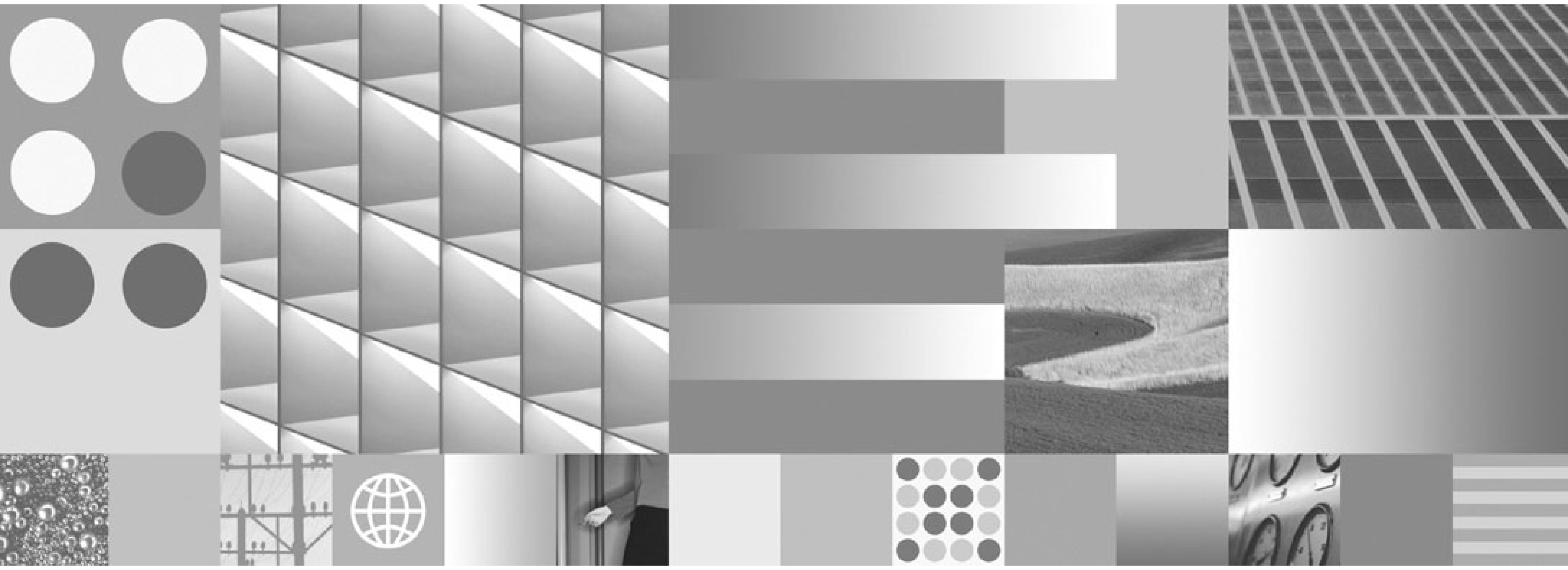
DB2 Version 9.5
for Linux, UNIX, and Windows

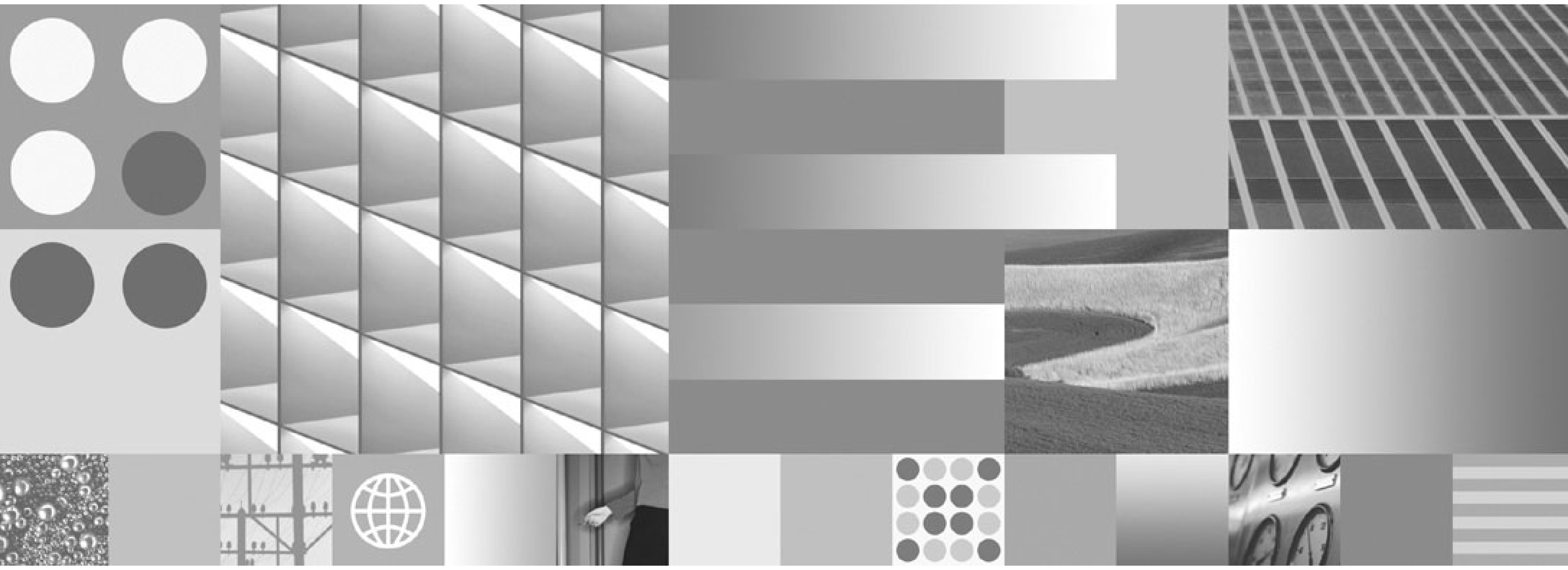
IBM



Workload Manager Guide and Reference
Updated March, 2008

SC23-5870-01

DB2 Version 9.5
for Linux, UNIX, and Windows



Workload Manager Guide and Reference
Updated March, 2008

SC23-5870-01

Note

Before using this information and the product it supports, read the general information under Appendix H, “Notices,” on page 297.

Contents

About this book

This book provides information on the DB2® workload management features and functionality that you can use to obtain a stable, predictable execution environment that meets your business objectives. Using 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.

Part 1. Introduction

Chapter 1. Introduction to 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.

Stages of workload management

Workload management has three clearly defined stages: 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, starting with understanding your goals. In the grocery store example described in Chapter 1, “Introduction to 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 CPU resources. 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.
- 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 stages:

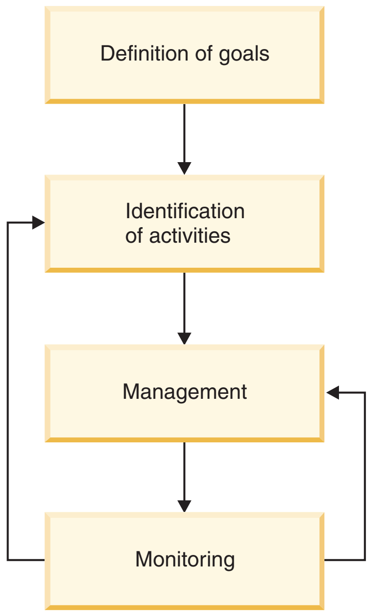


Figure 1. Stages of workload management

Identification stage of workload management

The first stage of implementing a workload management solution is to identify the work that runs on the data server.

Different methods can be used to identify database activities. For example, you can identify activities by their source (that is, who submitted the work). The following figure shows a number of different sources of work, coming from different users, groups, and applications. Activities can also be identified by type.

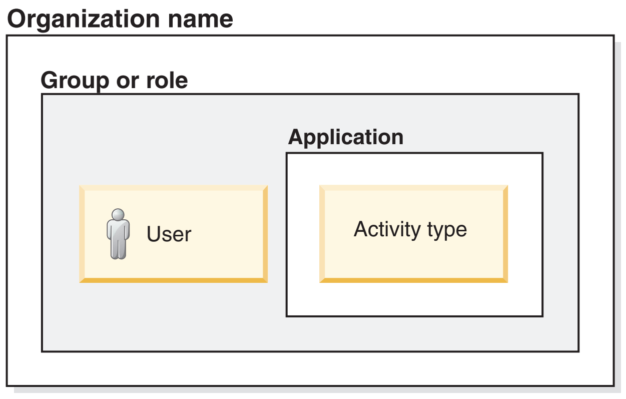


Figure 2. Various sources and types of database activities on a data server

To implement identification, two new database objects are provided: the workload and the work class objects.

Workloads

A *workload* is an object that is used to identify incoming work based on its source so that it can later be managed. The source is determined using the attributes of the database connection under which the work is submitted.

The connection attributes are evaluated when the connection is established, and the connection is assigned to a workload definition at which time a new occurrence of that workload, referred to as a workload occurrence, is started. If any of the connection attributes change during the life of that connection, the workload assignment is reevaluated at the start of the next unit of work after the change and, if a new workload definition is to be assigned, the old workload occurrence for the previous assigned workload is ended and a new occurrence is started for the newly assigned workload definition. While each connection is assigned to one and only one workload at any one time, it is possible for there to be multiple connections assigned to the same workload at the same time resulting in the concurrent existence of multiple workload occurrences related to that definition.

For the connection attributes supported for a workload, see Chapter 3, "Workloads," on page 39.

Workload reevaluation occurs at the beginning of each unit of work in the event that the value of a connection attribute or the workload definition itself changes during the unit of work. This reevaluation might result in the connection being associated with a new workload, creating a different workload occurrence.

First example of creating a workload

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

This command creates a workload object called REPORTING. All connections with an application name attribute of `Accounts` are assigned to this workload and they run in the `Marketing` service class, as shown in the following figure:

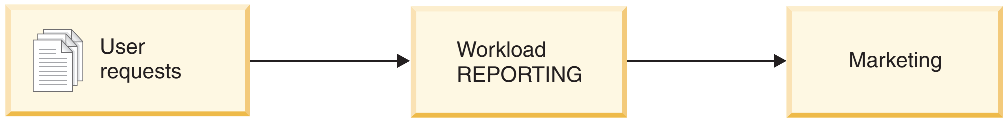


Figure 3. The REPORTING workload

Second example of creating a workload

```
CREATE WORKLOAD "SUMMARY" SESSION_USER_GROUP('Deptmgr') APPLNAME('Accounts')
SERVICE CLASS HumanResources
```

This command creates a workload object called SUMMARY. All connections with an application name `Accounts` and belonging to the session user group `Deptmgr` whose session user authorization ID belongs to the `DeptMgr` group map to the SUMMARY workload and are assigned to run in the `HumanResources` service class, as shown in the following figure:

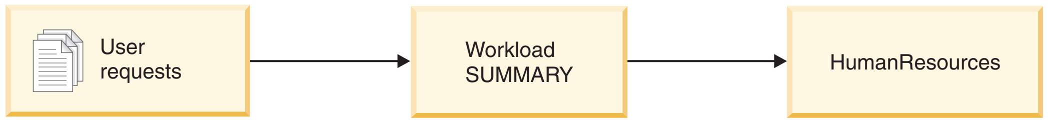


Figure 4. The SUMMARY workload

Connections that are not assigned to a custom-defined workload during workload evaluation will be assigned to the default workload. The default workload ensures that all database connections are associated with a workload.

Work classes

In addition to using connection attributes that focus on the source of the activities, you can identify activities based on the type of work through the creation of an optional *work class*.

A *work class set* represents a common definition of work types that can be used in different parts of the DB2 data server in conjunction with a work action set.

Table 1. Work types

Work type	Description
READ	Includes all SELECT and XQuery statements where only data is being fetched (that is, tables are not being updated)
WRITE	Includes all statements that modify data content on the data server (that is, INSERT, UPDATE, DELETE, and MERGE, even if they are imbedded in a SELECT statement)
CALL	Includes all invocations of procedures using a CALL statement
DML	Combines work found in the READ and WRITE work types
DDL	Includes statements that create or modify database objects (that is, CREATE, ALTER, DROP, COMMENT, DECLARE GLOBAL TEMPORARY TABLE, REFRESH TABLE, RENAME, GRANT, REVOKE, SET INTEGRITY)
LOAD	Includes all work initiated by the load utility on the data server
ALL	Includes all types of work

Work classes also introduce the ability to use predictive elements in the identification for DML work (or READ and WRITE statements). Predictive elements are very 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:

Table 2. Characteristics for predictive identification

Predictive element	Description
Estimated cost	Uses the estimated cost available from the DB2 compiler to include DML within a given timeron range (for example, create a work class set for all large queries with an estimated cost over 1 000 000 timerons)
Estimated cardinality	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)

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.

You can also use work actions to partly define how to manage the activities in the work class. For a work class to be active, you must define at least one work action for it. For more information, see the description of work action sets in "Management stage of workload management."

Management stage of workload management

Following the identification of the work, the next stage is the active management of the work, where you assign resources and impose controls on that work.

Service classes

The purpose of a service class is to define an execution environment in which the work can run. This environment can include available resources and various execution thresholds

When you define a workload, you must indicate the service class where work associated with that workload runs. The default workload also exists, which ensures that all data server work is running inside a service class.

Prioritization and resource control

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

Table 3. Resource control afforded by service classes

Control	Description
Agent priority	This control sets a CPU priority level for the agent threads running in a service class. This priority flows through to the operating system as a relative (delta) priority to other threads and processes running in the data server. **Note:** Not active when outbound correlator is in use.
Prefetch priority	This control assigns a priority to the prefetch requests, which affects the order in which they are addressed by the data server.
Outbound correlator	This control allows a workload to have some of its resources controlled by an external workload manager such as the AIX® Workload Manager (currently, the only supported external workload manager). 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 the AIX Workload Manager, additional capabilities are available. You can use the AIX Workload Manager to control the amount of CPU allocated to each service class. Options include setting a minimum, maximum, or relative proportion share of CPU for each service class.

Service subclasses

Although the service superclass is the highest tier for work, activities only run in service subclasses. This distinction is important to note. 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 addition subclasses in a service class as you require them to further isolate work or resources.

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 workload management configuration using workloads and service classes:

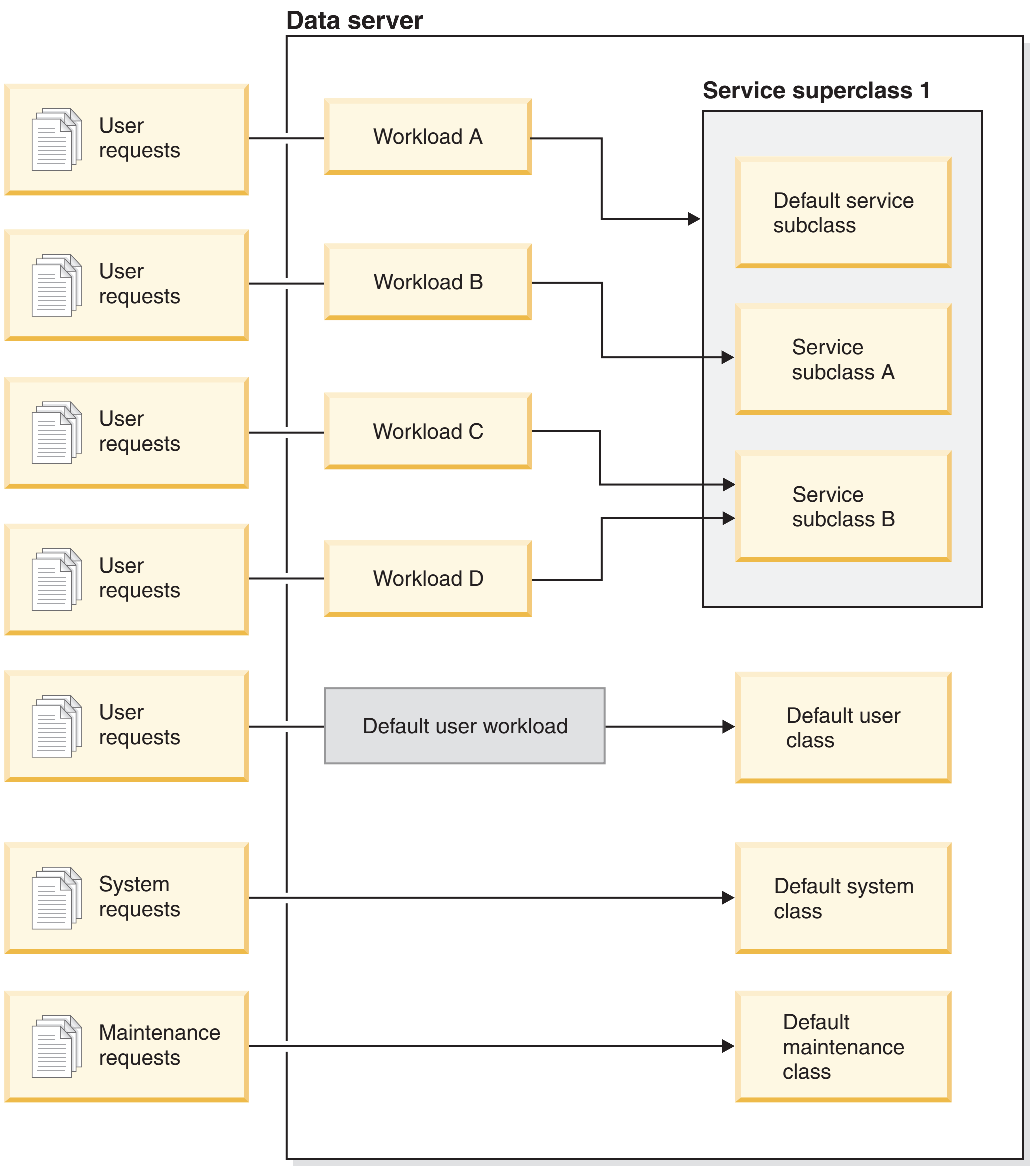


Figure 5. A custom workload management configuration using workloads and service classes

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 and DB2-driven maintenance requests (such as an automatic RUNSTATS from the health monitor) that run under a default maintenance service class.

Thresholds

Resource controls are one way to try to maintain a constant state of well-being on the data server: well-being meaning that your defined goals are being met. At times, work comes in that might exceed your normal expectations (a query that is returning hundreds of thousands of rows, for example), consuming valuable resources at the expense of all of the other work running on the system.

You can create threshold objects to maintain order in the system, and to catch work that behaves abnormally. The following tables provide information about the different thresholds that you can define. The first set of thresholds deals with activity limits, where the controls pertain to the impact that an activity can have on how the data server is running. Excess time, abnormally high volumes of data returned, and abnormally high amounts of resources consumed are warning flags that potentially troublesome activities could be using up resources that they are not supposed to.

Table 4. Activity limit thresholds

Threshold	Description
ACTIVITYTOTALTIME	Controls the amount of time that any given activity can spend from submission to completion in the DB2 data server (execution time and queue time). Use to detect jobs that are taking an abnormally long time to complete.
CONNECTIONIDLETIME	Controls the amount of time that a connection sits idle, not working on behalf of user requests. Use to detect inefficient use of data server resources and application wait conditions.
ESTIMATEDSQLCOST	Controls DML activities that the DB2 optimizer determines to have a large estimated cost. Use to predict potentially resource-heavy SQL before it starts executing on the system and identifying poorly written SQL.
SQLROWSRETURNED	Controls the number of rows returned when executing SQL. Use to identify when the amount of data exceeds a reasonable volume.
SQLTEMPSPACE	Controls the amount of temporary table space a given activity can consume on a partition. Use to prevent situations where a few errant SQL statements can use up a disproportionate amount of temporary space, impeding the progress of other work.

The next set of thresholds deals with concurrency control and is meant to look for cases where you need to consider limiting the number of certain activities running at the same time to reduce their impact on the data server.

Table 5. Concurrency limit thresholds

Threshold	Description
CONCURRENTWORKLOADOCCURRENCES	Controls the number of active occurrences of a workload that can run on a coordinator partition at the same time. Use to control the spread of connections from a specific source.
TOTALDBPARTITIONCONNECTIONS	Controls the number of database connections to a given partition that can be established at the same time. Use to prevent a given partition from becoming overloaded.
TOTALSCPARTITIONCONNECTIONS	Controls the number of database connections to a given partition for work executing within a given service class at the same time. Similar to the total database partition connections but more granular because the connection is linked to a service class.
CONCURRENTWORKLOADACTIVITIES	Controls the number of individual activities that can run within a workload occurrence. Use to limit work within an individual workload occurrence.
CONCURRENTDBCOORDACTIVITIES	Controls the number of concurrent activities in the domain that the threshold is associated with (database, work action, service superclass, or service subclass).

The kind of actions that can be taken when a threshold is violated depends on the threshold itself.

Collect data
: When some thresholds are violated, some form of data is collected. By default, the fact that a threshold was violated is recorded in an activated threshold violations event monitor. You might want more detail, though, so on each threshold definition you can request that more data be captured in an activated activities event monitor, such as information about individual activities, the statement text, the compilation environment, and even the input data values.

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.

Continue execution
: In some situations, stopping the execution of the activity is too harsh a measure. A preferable response is to allow the activity to continue to run and 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.

Work action set

As described in "Work classes" on page 7, you can define work classes to represent activities of a certain type (such as LOAD activities or READ activities). A work action provides an action that can be applied to a work class. A work action set can contain one or more work actions that can be applied to activities in either a specific superclass or to the database as a whole. For a work class to be active and have activities assigned to it, there must be a work action defined for the work class.

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 actions. 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 83.

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 class, preventing execution, collecting activity or aggregate activity data, and counting the activities. Typically, the work action set maps an activity to a service subclass and has thresholds defined on the subclass to help manage the activity.

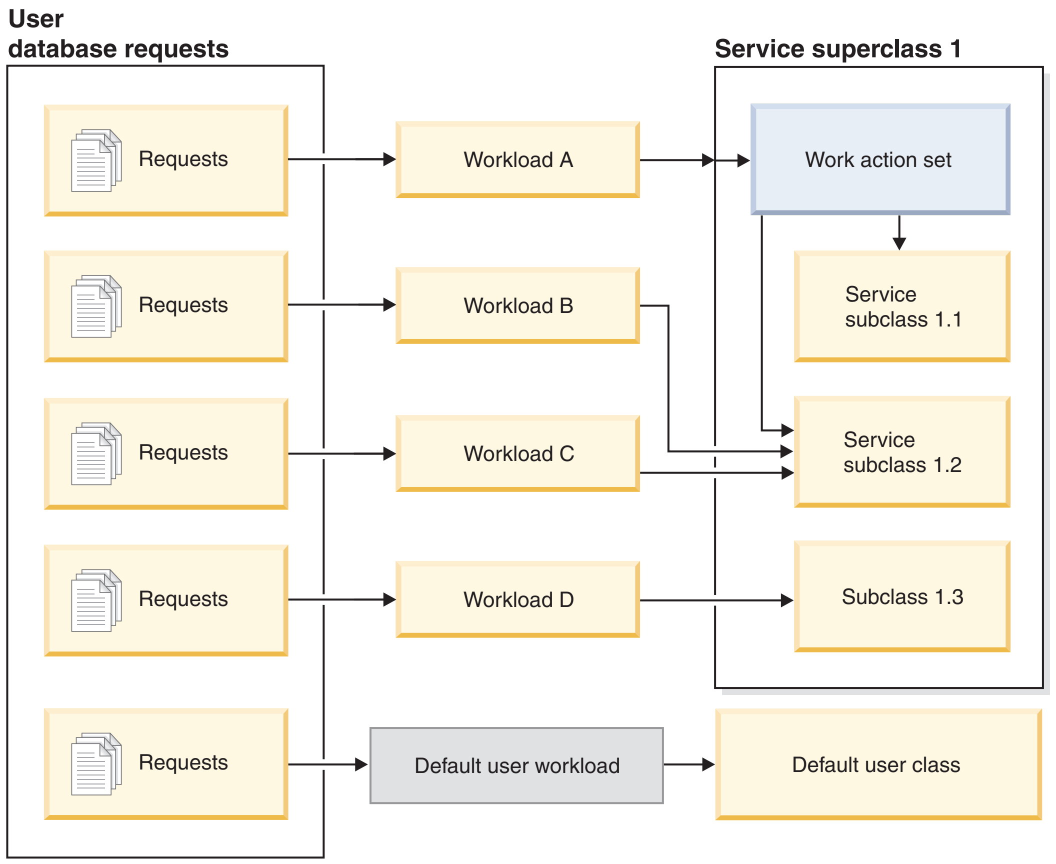


Figure 6. Work action set mapping for a service superclass

Monitoring stage of workload management

The third stage of workload management is monitoring. While this is described as the last stage of workload management, monitoring is ongoing.

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.

Realtime monitoring

With real-time monitoring, you can determine what is happening on your system in response to performance issues or problem reports.

Realtime monitoring data includes statistics that represent the current activity on the system that can help determine usage patterns and resource allocation and identify problem areas.

The method for accessing the real-time monitor data is through DB2 table functions. Table functions provide access to a set of data that exists inside a DB2 database (such as the workload management statistics) as a virtual DB2 table against which you can execute a SELECT statement. This offers you the ability to write applications to query data and analyze it just as if it were any physical table on the data server.

Some table functions return sets of information about the work that is currently executing on a system. This information about work is available at various levels:

Table 6. Table function information available for running work

Objects for which information is collected	**Description**
Workload occurrence	The WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function returns a list of workload occurrences, across database partitions, 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.
Service class agents	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.
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, DDL), and the time at which the activity started.
Activity details	The WLM_GET_ACTIVITY_DETAILS table function returns details about an individual activity. One detail returned is the activity type; depending on that type, a set of additional data is returned. For example, for SQL activities, information is available about the statement text, package data, cost estimates, and rows returned or modified. Details about the isolation level and CPU times are also available.

General statistical information is also available at a number of different levels, including the ones described in the following table:

Table 7. Statistical information available for a workload management solution

Objects for which statistics are returned	**Description of statistics returned**
Super service classes	The WLM_GET_SERVICE_SUPERCLASS_STATS table function shows summary statistics across database partitions at the service superclass level: namely, high water marks for concurrent connections (useful when determining peak workload activity).

Table 7. Statistical information available for a workload management solution (continued)

Objects for which statistics are returned	Description of statistics returned
Service subclasses	The WLM_GET_SERVICE_SUBCLASS_STATS table function shows summary statistics across database partitions at the service subclass level (all activities run in service subclasses). Statistics include numbers of completed activities and average execution times (useful when looking at general system health and distribution of activities across service classes and database partitions).
Workloads	The WLM_GET_WORKLOAD_STATS table function shows summary statistics across database partitions at the workload level. This includes high water marks for concurrent workload occurrences and numbers of completed activities (useful when monitoring general system health or drilling down to identify problem areas).
Work action sets	The WLM_GET_WORK_ACTION_SET_STATS table function shows summary statistics across database partitions at the work action set level: namely, the number of activities of each work class that had the corresponding work actions applied to them (useful for understanding the effectiveness of a work action set and understanding the types of activities executing on the system).
Threshold queues	The WLM_GET_QUEUE_STATS table function shows summary statistics across database partitions for the queues used for their corresponding thresholds. Statistics include the number of queued activities (current and total) and total time spent in a queue (useful when querying current queued activity or validating that a threshold is correctly defined: 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).

Statistics are only useful if the time period during which they are collected is meaningful. Collecting statistics over a very long time, and any 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, there is the ability to 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 be a great help in kick starting or validating the identification stage of workload management by helping to isolate sources of activities that you can use to create workloads and the service classes to which you can assign them.

Example: Using workload management table functions

A large amount of data is available through 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 migrate to DB2 9.5, 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.

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 database partitions (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.
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 partitions 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.
   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 like these, it might make sense to define some thresholds to identify and prevent potentially damaging behavior in the future.

Historical monitoring

In addition to the table functions, DB2 workload manager 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 in your workload management configuration. 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.

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.

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.

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 details individual activities:

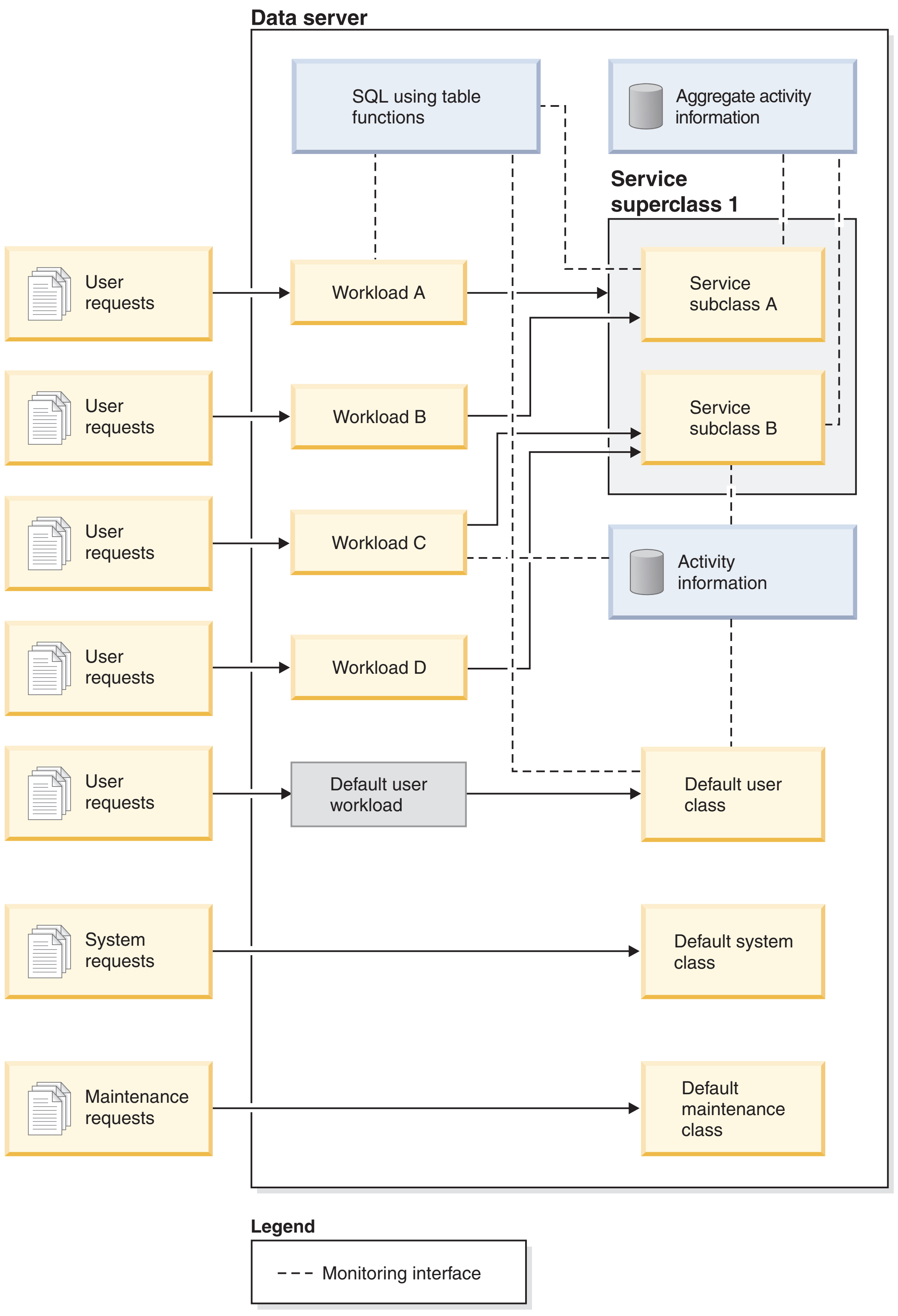


Figure 7. Workload management with monitoring

Activities

An activity is an individual piece of work that consumes database resource during its lifetime. The lifetime can span one or more database requests. A CALL statement or a cursor are examples of activities; an OPEN and FETCH request are examples of database requests that occur during the life of a cursor activity.

The data server recognizes the following activities for participation in a workload management configuration:

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

Work that is not classified as any of the previous activities above is still assigned to a workload (and the corresponding service class that is specified in the workload definition), based on the attributes of the connection that the work is submitted under. However, this work will not have thresholds or work actions applied to it.

In the context of activities, nesting refers to the situation where one activity invokes another. For example, a stored procedure is an activity. If the stored procedure contains a DML activity, that activity is a nested activity. If the stored procedure calls another stored procedure, the second stored procedure is a nested activity.

Workload management sample application

Comprehensive workload management has been integrated into Version 9.5, 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 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 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 allow 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 allow 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.

Part 2. Identification and management

Chapter 2. Service classes

All database requests are executed in a service class. The service class is the primary point for resource assignment to all database requests. Service classes are also used for monitoring and controlling sets of database activities in a database.

Every DB2 service class is either a service superclass or a service subclass in a service superclass. Each database can have multiple service superclasses and each service superclass can have multiple service subclasses. Service subclasses can only belong to one service superclass. The resources of the superclass are shared by all subclasses in it.

You set up service classes to organize activities in the database so that you can meet the performance objectives of your organization. Without service classes, requests cannot be organized into recognizable, logical groupings, as is shown in the following figure.

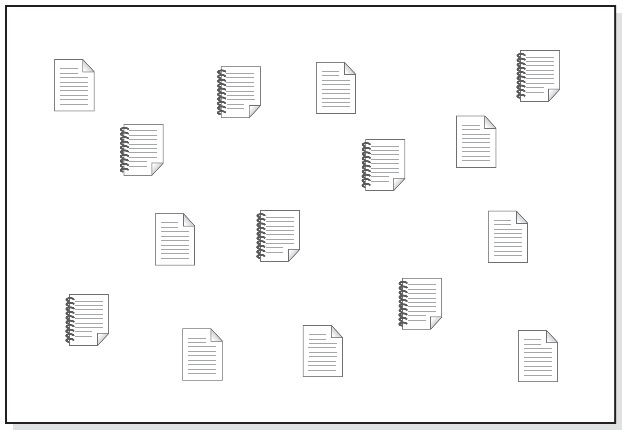


Figure 8. Unorganized work

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.

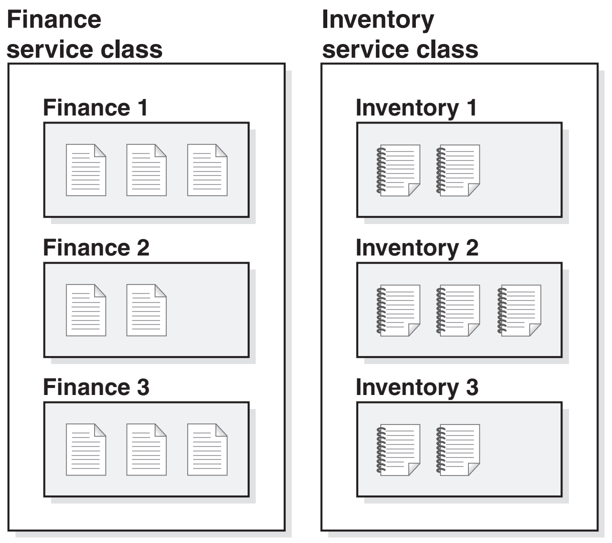


Figure 9. Work organized by service classes

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 allows 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.

All work actually runs in the service subclasses and a default service subclass is automatically created when a service superclass is created. That default subclass is what is used by any work mapped to (and left in) the service superclass. The service superclass acts as a common background for all of its service subclasses. Except for histograms and the COLLECT ACTIVITY DATA options, a service subclass inherits the attributes of its service superclass, unless otherwise specified.

Because different work has different priorities, when using service classes, you can control a number of characteristics of that service class. For example:

- You can set the I/O page prefetcher priority to be used for work in the service class.
- You can set the agent CPU priority for all agents in the service class.
- You can control resource usage in the service class by applying constraints on the work running in a service class by using different types of thresholds.

You use workloads to assign work to service superclasses. You can also assign work to service subclasses in a service superclass by using workload definitions or work actions.

You can create service classes by using the CREATE SERVICE CLASS statement. You can alter service classes by using the ALTER SERVICE CLASS statement. You can drop service classes by using the DROP SERVICE CLASS statement.

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

Default service superclasses and subclasses

When you install or migrate to DB2 Version 9.5 or later, each new database or migrated 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.

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:

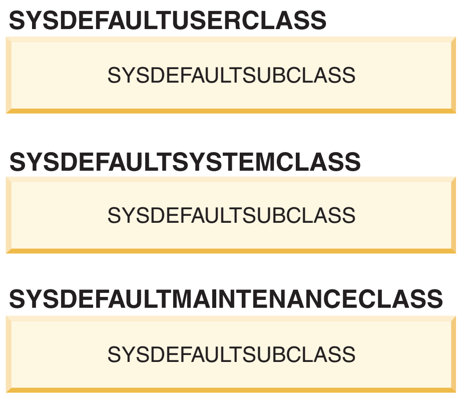


Figure 10. Two-tier service class hierarchy

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)
After installing or migrating to DB2 Version 9.5, by default, all 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
- Query Patroller (QP) connections
- 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)
- Connections performing system level tasks

A Query Patroller connection is an internal connection to the DB2 data server issued by the QP controller (the server component of QP) when QP is started. This connection is established as QP is starting up, and after QP has successfully started, the connection is mapped to the default system service superclass. Whilst QP is starting up, the connection may temporarily be mapped to another service class as part of the normal workload mapping process. During this period, the connection is subject to all controls and thresholds of the service class it is temporarily mapped to.

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. If the workload occurrence is assigned to a service superclass, any work submitted for that workload occurrence can be reassigned to a user-defined service subclass in that service superclass by a work action (applied to a work class) if a work action set is defined for the service superclass (that is, not the default service subclass).

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 implementation 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.

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.

Note: Only the coordinator agent does service superclass mapping for the connection. If the coordinator agent spawns subagents, the subagents inherit the superclass mapping of the coordinator agent.

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.

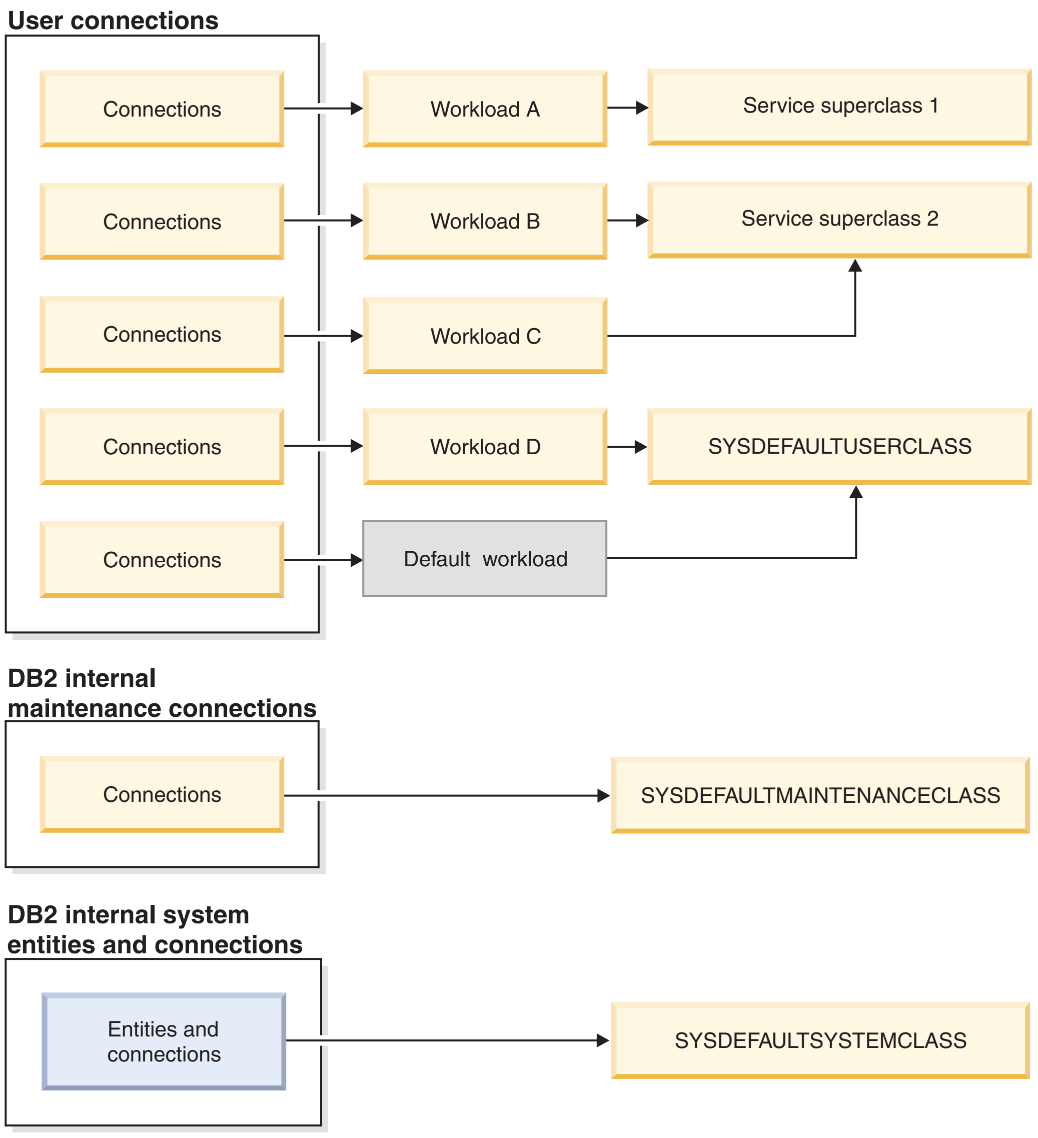


Figure 11. Mapping of database connections to a service superclass

If you have a more complex workload management 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 maps an activity to a service subclass, that activity is not affected by any work action in a work action set that is applied to the 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 connections 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 77

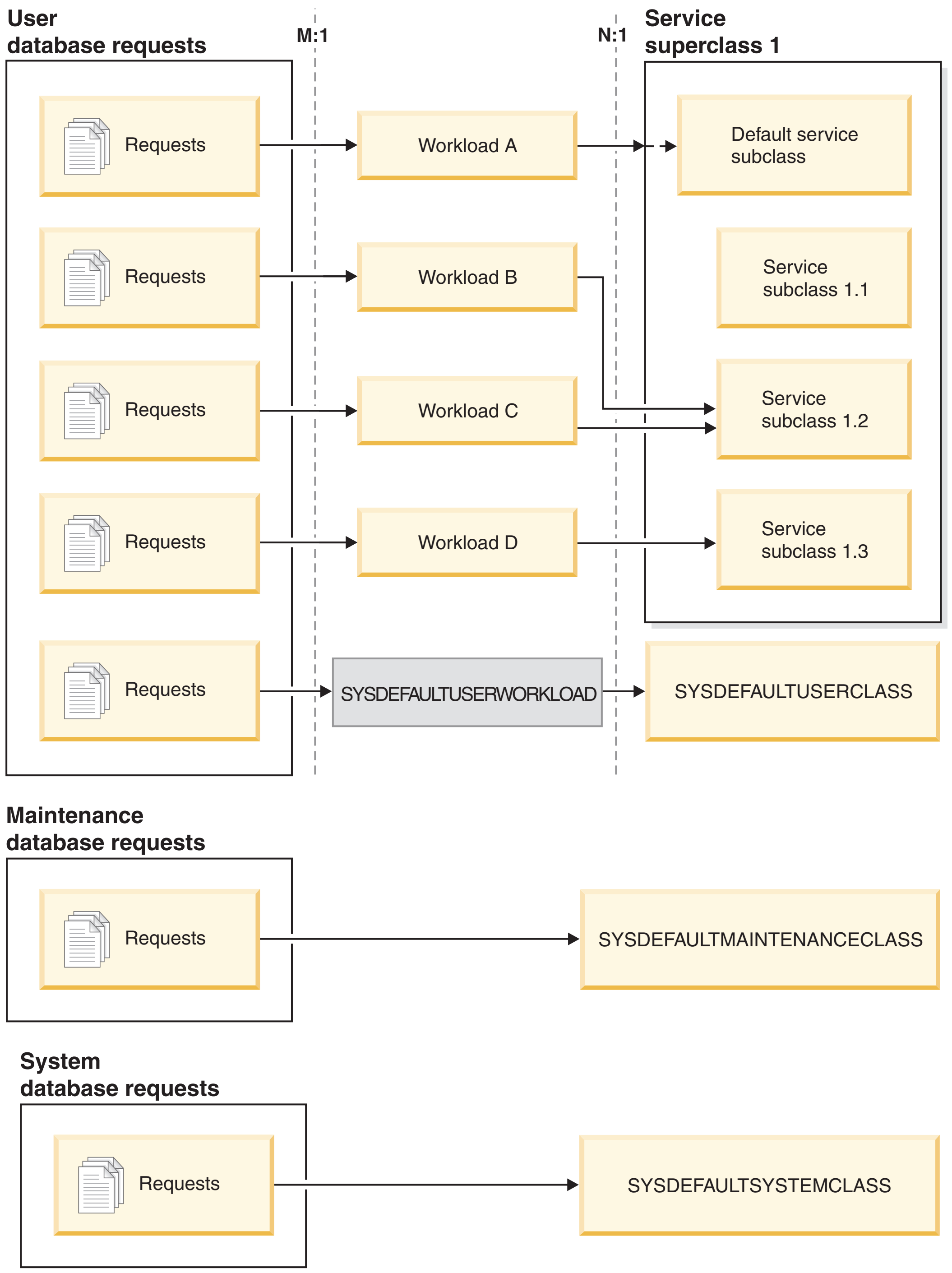


Figure 12. Database connections being mapped to a service superclass

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.

CPU priority and DB2 service classes

With DB2 service classes, every service class can be associated with a relative agent priority. This priority is set for all agents that work in a service class and is relative to the agent priority of all other DB2 agents. If you do not specify the agent priority value for a service class, all agents in that service class have the same priority as all other DB2 agents.

Setting the agent priority for a DB2 service class only adjusts the priority of agents running in that service class for new work that enters. Other non-agent threads running in the service class do not use the agent priority value specified for the service class.

If you are integrating DB2 service classes with the AIX workload manager, you can use the AIX Workload Manager to specify the CPU priorities to be used for the operating system class, then have the DB2 service class inherit this value through the OUTBOUND CORRELATOR value of the DB2 service class. In this situation, the CPU priority you specify using the operating system workload manager will control the priority for agents that run in the DB2 service class, and any service class agent priority setting is ignored.

The **agentpri** database manager configuration parameter sets the absolute CPU priority of all agents in a DB2 instance to a fixed value. When the absolute priority is set for an agent, its relative priority cannot be altered. For this reason, **agentpri** is not compatible with service class agent priority or integration with the AIX Workload Manager. If **agentpri** is set to a non-default value, the service class agent priority and AIX Workload Manager have no effect on the priority of agents. You should not use this deprecated configuration parameter in a workload management configuration.

Note that on AIX, the instance owner must have CAP_NUMA_ATTACH and CAP_PROPAGATE capabilities to set a higher relative priority for agents in a service class using AGENT PRIORITY. To grant these capabilities, logon as root and run the following command:

```
chuser capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE
```

Service class prefetch priority

Prefetchers retrieve data from disk and store this data in buffer pools so that it can be quickly accessed by agents. In DB2 workload management, 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.

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.

Mapped
The connection is mapped to a workload and has joined a service superclass. The connection can now submit activities for execution.

Transient
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.

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

States of an activity

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

Initializing
The activity was created and is being prepared for execution.

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

Note: On the AIX operating system, if a queued activity receives SQL4297N, ensure that the DB2 client and data server have the following APAR installed:

- For AIX 5.3, IY89429
- For AIX 5.2, IY89387

Executing
The activity is executing.

QP Queued
The activity is queued by Query Patroller.

Terminating
The activity is being terminated.

System-level entities not tracked by service classes

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 tracked 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 tracked by service classes either. Because instance attachment agents and gateway agents do not work in a database, they are not tracked by service classes.

The following list is a partial list of entities that do not work within a database and are not tracked by service classes:

- DB2 system controllers (db2sysc)
- IPC listeners (db2ipccm)
- TCP listeners (db2tcpcm)
- 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

Working with service classes

Creating a service class

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

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

Also see the following topics for other prerequisites:

- Appendix A, “Workload management DDL statement considerations,” on page 267
- Naming rules

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.
- Optional: Specify the agent priority for the service class. When the agent priority is set to DEFAULT, agents in the service class are assigned the same priority that the operating system assigns all DB2 threads. If you set the AGENT PRIORITY parameter to a value other than DEFAULT, the agent threads are set to a priority equal to the default priority, plus the value set when the next activity begins. For example, if the default priority is 20 and you set agent priority to -10, the resulting agent priority is set to 20 + (-10) = 10.

  An agent priority of DEFAULT evaluates to a numeric value of -32768.

  On Linux® and UNIX®, the valid values are -32768, -20 to 20 (a negative value indicates a higher relative priority). On Windows-based platforms, the valid values are -32768, -6 to 6 (a negative value indicates a lower relative priority)
- Specify the outbound correlator string if you want to associate the DB2 service class to an AIX service class. A null value indicates no external WLM service class association.

  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.

  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 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.
- Activity data to collect. When activity data collection is enabled, information about an activity is sent from the coordinator partition to the applicable event monitor at the end of the activity. If you want, you can write data to the event monitor from all database partitions on which the activity ran, including information about the statement that was run, its compilation environment, and any applicable input data values. You can also specify that no data activity 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.
- 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 on histograms and histogram templates, see "Histograms in workload management" on page 117.
- 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 may 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.

To alter a service class, you require DBADM, SYSADM, or SYSCNTRL authority.

See Appendix A, "Workload management DDL statement considerations," on page 267 for more information about prerequisites.

Activities that have already acquired resources and are running are not affected by the ALTER statement. These activities will hold their resources and run until completion. However, if a subagent request is sent to a remote database partition 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 8. Differences between the views of a coordinator agent and subagent of an altered service class

Event order	Connection 1	Connection 2
1	Coordinating agent sends DSS request to remote partition (prefetch priority of service class was previously set to MEDIUM)	
2		ALTER SERVICE CLASS issued; set prefetch priority to HIGH
3		COMMIT is issued (the altered service class property is committed at the catalog partition and loaded to memory at all database partitions)

Table 8. Differences between the views of a coordinator agent and subagent of an altered service class (continued)

Event order	Connection 1	Connection 2
4	Remote subagent: receive DSS request. At this time, the subagent sees the new prefetch priority of HIGH for the service class definition	

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.

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 agent priority for the service class. When the agent priority is set to DEFAULT, agents in the service class are assigned the same priority that the operating system assigns all DB2 threads. If you set the AGENT PRIORITY parameter to a value other than DEFAULT, the agent threads are set to a priority equal to the default priority, plus the value set when the next activity begins. For example, if the default priority is 20 and you set agent priority to -10, the resulting agent priority is set to 20 + (-10) = 10.

     An agent priority of DEFAULT evaluates to a numeric value of -32768.

     On Linux and UNIX, the valid values are -32768, -20 to 20 (a negative value indicates a higher relative priority). On Windows-based platforms, the valid values are -32768, -6 to 6 (a negative value indicates a lower relative priority)
   - 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 outbound correlator string if you want to associate the DB2 service class to an AIX service class. A null value indicates no external WLM service class 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.

  If a service superclass uses an outbound correlator, the agent priority of the service superclass must be set to default.

  If a service subclass uses an outbound correlator (either explicitly as part of the service subclass definition or implicitly through inheritance from the service superclass), the agent priority of the service subclass must be set to default.
- Activity data to collect. When activity data collection is enabled, information about an activity is sent from the coordinator partition to the applicable event monitor at the end of the activity. If you want, you can write data to the event monitor from all database partitions on which the activity ran, including information about the statement that was run, its compilation environment, and any applicable input data values. You can also specify that no data activity 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.
- 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 on histograms and histogram templates, see “Histograms in workload management” on page 117.

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.

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

See Appendix A, “Workload management DDL statement considerations,” on page 267 for more information about prerequisites.

Following are the restrictions for dropping a service class:

- 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 cannot be dropped if any of the following conditions apply to the service class:

  - It is enabled
  - It is referenced by any workload, work action or threshold
  - Any connection or activity is currently mapped to the service class

To drop a service class:

1. 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.

   Activities already running under the service class will continue to run. You can list agents that are currently mapped to a service class 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. Use the DROP WORKLOAD statement to drop all workloads associated with the service class. Issue a COMMIT statement after dropping each workload.
3. 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, drop that work action set using the DROP WORK ACTION SET statement. 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 keyword of the ALTER WORK ACTION SET statement. Alternatively, you can also 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.
4. Use the DROP THRESHOLD statement to drop all thresholds associated with the service class you want to drop. Issue a COMMIT statement after dropping each threshold.
5. 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.
6. Commit your changes. When you commit your changes the service class is removed from the SYSCAT.SERVICECLASSES view.

Chapter 3. Workloads

A workload is an entity that you define that identifies submitted database work based on its source, according to the database connection attributes, so that it can later be managed. Using workloads, you can assign work to a service superclass or to a service subclass in a service superclass.

A workload object consists of the following items:

- A workload name that is unique in the database.
- A unique integer identifier for the workload, generated and used internally by the data server.
- Database connection attributes that must be satisfied for a database connection or application to be associated with the workload.
- Workload attributes, including the following items:
  - The name of the DB2 service class to which the workload is to be assigned; if you don't specify a service class name, the workload is mapped to the default user service class, SYSDEFAULTUSERCLASS.
  - A value that indicates whether an occurrence of the workload is allowed to access the database. By default, workload occurrences can access the database. For more information, see "Allowing occurrences of a workload to access the database" on page 49 and "Disallowing occurrences of a workload from accessing the database" on page 50
  - A value that indicates whether the workload is disabled. The default is enabled. For more information, see "Enabling a workload" on page 50 and "Disabling a workload" on page 51.
- The evaluation order or position of the workload relative to other workloads on the data server. For more information, see "Workload assignment" on page 42.

You can create workloads by using the CREATE WORKLOAD statement. You can alter workloads by using the ALTER WORKLOAD statement. You can drop workloads by using the DROP WORKLOAD statement.

You can view your workloads by querying the SYSCAT.WORKLOADS view. 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.

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

Table 9. Connection attributes in a workload definition

Connection attribute	Description
Application name	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 **appl_name** monitor element for more information.

Table 9. Connection attributes in a workload definition (continued)

Connection attribute	**Description**
System authorization ID	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.
Session authorization ID	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 using the SET SESSION AUTHORIZATION statement.
Group of session authorization ID	The groups to which the current session user belongs.
Role of session authorization ID	The roles granted to the current session user. For more information, see: • Roles • GRANT ROLE statement • REVOKE ROLE statement
Client user ID	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 the sqleseti (set client information) API
Client application name	The application name from the client information as set in the CURRENT CLIENT_APPLNAME (or CLIENT APPLNAME) special register. The client application name is equivalent to the value shown in the `TP Monitor client application name` field in the system monitor output. You can change the value of the client application name by using the sqleseti API.
Client workstation name	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 the sqleseti API.
Client accounting string	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.

In a three-tier client/server environment, the database connection is established by the application server that is working on behalf of the clients. The application server can use the sqleseti API to pass client information to the DB2 data server; otherwise, only the information about the application server is passed, and that information is likely to be the same for all client requests that are routed through this application server. By specifying client information attributes such as the client user ID, client application name, client workstation name, and client accounting string in the workload definition, you can assign users running from different clients to different workloads (and to different service classes).

At the beginning of the first unit of work, the database connection is assigned to a workload if its connection attributes match the connection attributes that you specified in the workload definition. The database connection is reassigned to a different workload at the beginning of the next unit of work if a connection attribute changes. For more information, see “Workload assignment” on page 42

SYSDEFAULTUSERWORKLOAD is the default workload. If the connection attributes do not map to any workload that you defined, the database connection is assigned to SYSDEFAULTUSERWORKLOAD, and the work is executed in the SYSDEFAULTUSERCLASS service class by default. This situation occurs in a new database or a newly migrated database because no workloads other than the default workloads exist.

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 42.

A workload occurrence is a database connection with attributes that match a workload definition. When the workload assignment completes, an occurrence of a workload is started in the data server and runs in the service class specified in the workload definition. This workload occurrence lasts until the connection terminates or a connection attribute changes, in which case workload reevaluation occurs at the beginning of the next unit of work. Workload reevaluation and reassignment occurs at unit of work boundaries. Therefore, a workload occurrence consists of one or more units of work in a database connection that is associated with a workload defined in the data server. More than one workload occurrence can run on the data server concurrently for each workload.

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 and executed by default in the SYSDEFAULTUSERCLASS service superclass. 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 25.

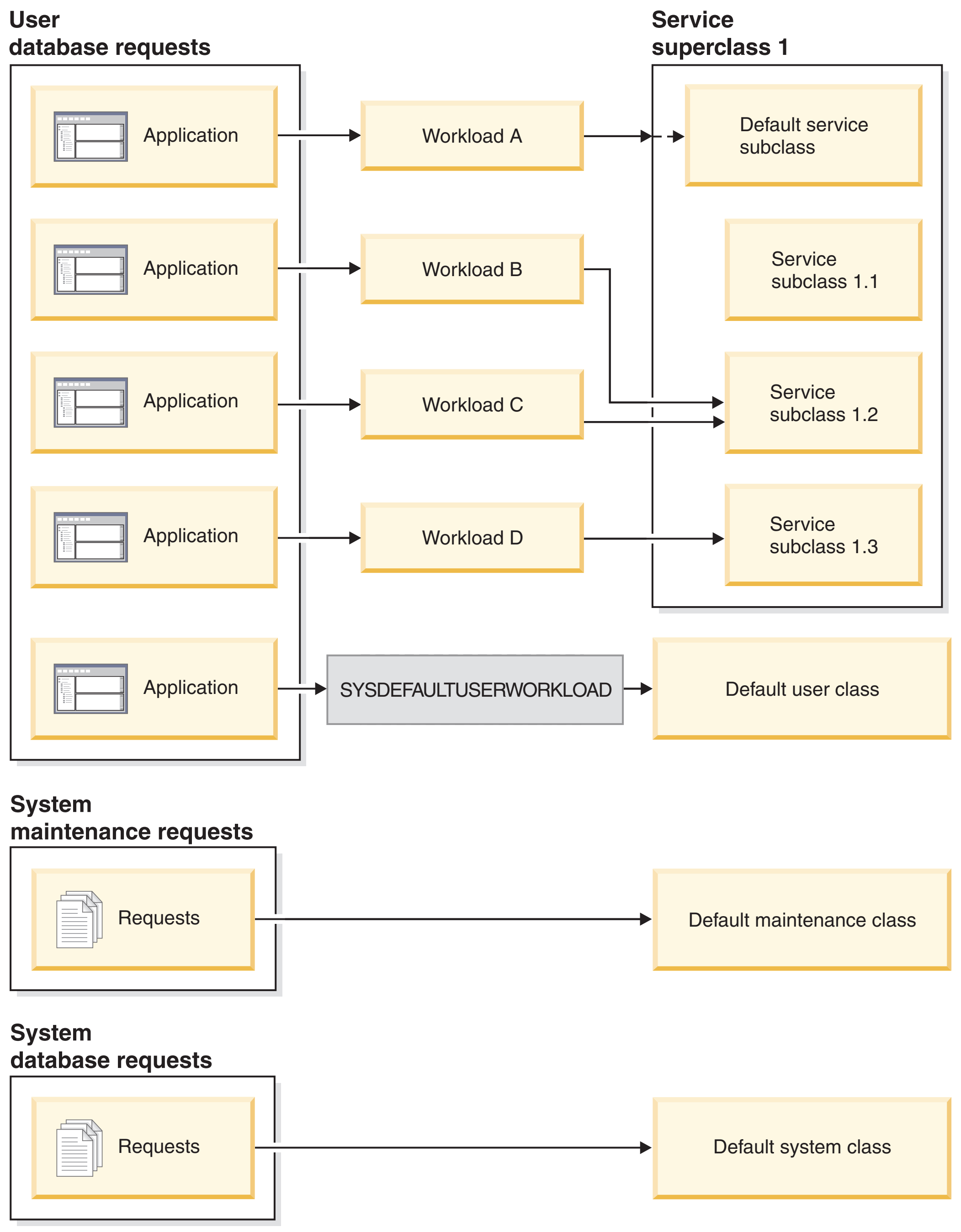


Figure 13. Service classes and workloads

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.

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 Chapter 3, "Workloads," on page 39 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 is still active. An activity is 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 disallowed access to the database. For more information, see “Disallowing occurrences of a workload from accessing the database” on page 50.
- 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 DBADM or SYSADM authority, you can assign your database connection to the SYSDEFAULTADMWORKLOAD workload, the default administrator workload. See “Assigning a connection to the default administration workload” on page 46 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, and the default administration workload, SYSDEFAULTADMWORKLOAD, are created at database creation time. You cannot drop them.

After DB2 installation or migration to Version 9.5 or later, all connections are assigned to the default workload, SYSDEFAULTUSERWORKLOAD. Connections that belong to this default workload are mapped to the default user service superclass, SYSDEFAULTUSERCLASS. You can remap connections from the default workload to user-defined workloads to use other user-defined service classes, if required. 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 the entry for the SYSDEFAULTUSERWORKLOAD workload in the SYSCAT.WORKLOADS view. See “Workload assignment” on page 42 for information on how to assign a connection to the SYSDEFAULTUSERWORKLOAD workload.

Table 10. SYSDEFAULTUSERWORKLOAD entry in SYSCAT.WORKLOADS

Column	Value	Modifiable using the ALTER WORKLOAD statement if you have DBADM or SYSADM access
WORKLOADID	1	No
WORKLOADNAME	SYSDEFAULTUSERWORKLOAD	No
EVALUATIONORDER	Second last one	No
CREATE_TIME	Timestamp of database creation	No

Table 10. SYSDEFAULTUSERWORKLOAD entry in SYSCAT.WORKLOADS (continued)

Column	Value	Modifiable using the ALTER WORKLOAD statement if you have DBADM or SYSADM access
ALTER_TIME	Timestamp of the last update of the workload definition	No (but the data server modifies this column when you update the workload definition)
ENABLED	Y	No
ALLOWACCESS	Y	Yes
SERVICECLASSNAME	SYSDEFAULTSUBCLASS	Yes
PARENTSERVICECLASSNAME	SYSDEFAULTUSERCLASS	Yes
COLLECTAGGACTDATA	N	No (reserved for future use)
COLLECTACTDATA	N	Yes
COLLECTACTPARTITION	C	Yes
EXTERNALNAME	NULL	No

For more information, see SYSCAT.WORKLOADS.

Only units of work submitted by a session user with SYSADM or DBADM authority can be assigned to the SYSDEFAULTADMWORKLOAD workload. This workload allows SYSADM and DBADM users to query the database and perform administrative or monitoring tasks in case the following events occur:

- The workload to which the administrator is assigned is not allowed 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 25 for more information.
- You can assign a connection to this workload only by using the SET WORKLOAD command. You can issue this command only from the CLP interface. For more information, see “Assigning a connection to the default administration workload” on page 46.

You can view the SYSDEFAULTADMWORKLOAD workload by querying the SYSCAT.WORKLOADS table. The following table shows the entry for the SYSDEFAULTADMWORKLOAD workload in the SYSCAT.WORKLOADS table:

Table 11. SYSDEFAULTADMWORKLOAD entry in SYSCAT.WORKLOADS

Column	Value	Modifiable using the ALTER WORKLOAD statement if you have DBADM or SYSADM access
WORKLOADID	2	No
WORKLOADNAME	SYSDEFAULTADMWORKLOAD	No
EVALUATIONORDER	Last one	No
CREATE_TIME	Timestamp of database creation	No
ALTER_TIME	Timestamp of the last update of the workload definition	No (but the data server modifies this column when you update the workload definition)
ENABLED	Y	No
ALLOWACCESS	Y	No
SERVICECLASSNAME	SYSDEFAULTSUBCLASS	No
PARENTSERVICECLASSNAME	SYSDEFAULTUSERCLASS	No
COLLECTAGGACTDATA	N	No (reserved for future use)
COLLECTACTDATA	N	Yes
COLLECTACTPARTITION	C	Yes
EXTERNALNAME	NULL	No

For more information, see SYSCAT.WORKLOADS.

Assigning a connection to the default administration workload

You can use the SET WORKLOAD command to assign a connection to the default administration workload, SYSDEFAULTADMWORKLOAD.

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

The default administration workload (SYSDEFAULTADMWORKLOAD) is a special DB2-supplied workload definition that is not subject to any DB2 thresholds. This workload is intended to allow the database administrator to perform their work or to take corrective actions, as required. As 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.

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 42.

Working with workloads

Creating a workload

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

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

See the following topics for more information about prerequisites:
- Appendix A, "Workload management DDL statement considerations," on page 267
- Naming rules

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 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 42.
   - 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 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 Chapter 3, "Workloads," on page 39. 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 allowed to access the database. By default, occurrences of this workload are allowed to access the database.

- A value that indicates whether the workload is enabled or disabled. By default, the workload is enabled.
- 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 and do not map the workload to run in a user-defined service subclass under the service superclass, the workload occurrences will run 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 do not want occurrences of the workload to run in the SYSDEFAULTSUBCLASS service subclass, you can map the workload for execution in a user-defined service subclass through the workload. You can also use a work action to map the workload to a different service subclass (for more information, see "Work actions and work action sets" on page 77).

2. Commit your changes. When you commit your changes the workload is added to the SYSCAT.WORKLOADS view. Committing the change causes a workload reevaluation 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.

After you create a workload, you might need to grant the USAGE privilege on it to one or more session users. (Session users with SYSADM 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 51.

Altering a workload

An ALTER WORKLOAD statement changes a workload in the catalogs.

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

See Appendix A, "Workload management DDL statement considerations," on page 267 for more information about prerequisites.

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 Chapter 3, "Workloads," on page 39. To see the connection attributes for a workload, query the SYSCAT.WORKLOADCONNATTR view.
   - A value that indicates whether an occurrence of this workload is allowed to access the database. By default, an occurrence of this workload is allowed to access the database. You cannot disallow the SYSDEFAULTADMWORKLOAD workload access to the database.

- 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. By default, a new workload is positioned last, which means that it is evaluated last, immediately before the default user workload is considered. You cannot specify the position of the SYSDEFAULTUSERWORKLOAD or the SYSDEFAULTADMWORKLOAD workload. For more information, see "Workload assignment" on page 42.
- The type of activity information to collect. By default, no information for activities associated with the workload is sent to an activities event monitor.

2. Commit your changes. When you commit your changes the workload is updated in the SYSCAT.WORKLOADS view. The committed change causes a workload reevaluation 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.

After you alter a workload, you might need to grant the USAGE privilege on it to one or more session users. (Session users with SYSADM 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 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 51.

Allowing occurrences of a workload to access the database

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

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

See Appendix A, "Workload management DDL statement considerations," on page 267 for more information about prerequisites.

When you disallow a workload access to the database, the data server still examines that workload when performing workload assignment. However, all occurrences of that workload are rejected with an error. To allow a workload to access the database:

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

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

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

Altering a workload to allow 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 allowed to execute. Previously, any occurrence of workload A would have been rejected with an error.

Disallowing 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 allowed to access the database. If you disallow the workload occurrence from accessing the database, an error is returned indicating that the workload occurrence is rejected.

To disallow a workload from accessing the database, you require DBADM or SYSADM authority.

See Appendix A, "Workload management DDL statement considerations," on page 267 for more information about prerequisites.

Disallowing 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. To disallow a workload from accessing a database:

1. Use the DISALLOW DB ACCESS option of the ALTER WORKLOAD statement, as shown in the following example:

   ```
   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.

Altering a workload to disallow 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.

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.

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

See Appendix A, "Workload management DDL statement considerations," on page 267 for more information about prerequisites.

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.

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.

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 The DB2 data server checks the connection attributes that you specify for a workload against the connection attributes of the current session. 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 unit of work is assigned to the default workload.

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

See Appendix A, “Workload management DDL statement considerations,” on page 267 for more information about prerequisites.

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.

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.

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 SYSADM and DBADM authorities implicitly have the USAGE privilege on all workloads.

To use the GRANT USAGE ON WORKLOAD statement, you require SYSADM or DBADM authority

See Appendix A, “Workload management DDL statement considerations,” on page 267 for more information about prerequisites.

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 42.

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 allows 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 without the RESTRICT option, the USAGE privilege on the SYSDEFAULTUSERWORKLOAD workload is granted to PUBLIC at database creation time. Otherwise, you must explicitly grant the USAGE privilege on this workload to non-SYSADM 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.

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:

   ```
   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 SYSADM and DBADM 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.

To use the REVOKE USAGE ON WORKLOAD statement, you require SYSADM or DBADM authority.

See Appendix A, “Workload management DDL statement considerations,” on page 267 for more information about prerequisites.

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

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:
   ```
   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.

Dropping a workload

Dropping a workload removes it from the database catalog.

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

See Appendix A, “Workload management DDL statement considerations,” on page 267 for more information about prerequisites.

To drop a workload:

1. Disable the workload by specifying the ALTER WORKLOAD statement. See “Disabling a workload” on page 51 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:
   ```
   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.

Chapter 4. Thresholds

You can use thresholds to detect resource misuse or the beginning of system overload.

Using thresholds, you can explicitly establish limits over the consumption of a specific resource. If a threshold is violated, a specified action can be triggered. The supported actions are as follows:

- Stop processing the activity that caused the threshold to be violated (STOP EXECUTION)
- Continue processing (CONTINUE)
- Collect information about the activity that violated the threshold

Whether an activity that violates a threshold is stopped or allowed to continue, you can collect detailed information about the activity. Information about the activity that violated the threshold is collected by the active ACTIVITIES event monitor when the activity completes execution.

Whether or not you collect information about the activity that violated the threshold, a threshold violations record is written to the active THRESHOLD VIOLATIONS event monitor when the threshold is violated.

Thresholds are classified into two general categories:

- Activity thresholds. An activity threshold applies to an individual activity. For example, the maximum activity total time threshold is an activity threshold because it limits the total amount of time a single activity can remain in the data server.
- Aggregate thresholds. An aggregate threshold sets a limit on a measurement that is computed across a set of multiple activities. For example, the maximum number of concurrent activities in a service class is an aggregate threshold.

Each threshold applies to a *domain*. The domain of a threshold defines the database object that the threshold operates on. Only activities taking place in the domain of a threshold may be affected by it. The threshold domains are as follows:

- Database
- Service superclass
- Service subclass
- Work action
- Workload

In each of these threshold domains, the threshold might be enforceable over a single workload occurrence, a database partition, or all the partitions of a database. This is known as the *enforcement scope* of the threshold.

Service class aggregate thresholds have one of two enforcement scopes: database and database partition. An example of an aggregate threshold that applies at the database partition level is the maximum number of concurrent connections for a service superclass on a partition. An example of an aggregate threshold that applies at the database level (that is, across all database partitions) is the maximum number of concurrent activities for a service class across all partitions.

Some thresholds have a built-in queue and are defined with two boundaries: a *threshold boundary* and a *queueing boundary*. These thresholds are known as *queueing thresholds*. The threshold boundary of a queuing threshold typically enforces some level of concurrency (such as the maximum number of concurrent activities), beyond which additional requests are queued. The queueing boundary defines the upper limit for the queue. Specifically, when an activity violates the threshold boundary of a queuing threshold, new work requests being tracked by that threshold are queued automatically in a first-in, first-out fashion, until the queue reaches the size specified by the queueing boundary. When the queue is full, the upper boundary is reached, and the action specified for the threshold is applied to any newly arriving work being tracked by that threshold. For example, an action of STOP EXECUTION causes the newly arriving work to be rejected.

It is also possible to define the upper boundary as being *unbounded*, in which case there is no upper limit to the size of the queue. In this situation, newly arriving work is added to the queue. If you define a hard limit for the upper boundary and define an action of CONTINUE as the threshold action, all newly arriving work that violates the threshold boundary is added to the queue, and the threshold behaves as if its queueing boundary were unbounded.

Different thresholds track different types of work. For example, a threshold might track SQL-based activities, utilities such as the load utility, connections, workload occurrences, and so on. Work that is of interest to a threshold is referred to as the *tracked work* for that threshold. For example, a threshold that is based on a number of timerons applies only to work that has a timeron value associated with it (in this situation, DML-based activities) and does not include other types of work such as DDL or utilities.

A threshold can be either predictive or reactive:

- The boundaries of a predictive threshold are checked before the tracked work starts running. To check whether a predictive threshold would be violated, the data server obtains usage estimates from the SQL compiler.
- The boundaries of a reactive threshold are checked while a tracked piece of work 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.

Thresholds do not apply to all statements. For example, they do not apply to COMMIT, ROLLBACK, SAVEPOINT, and ROLLBACK to SAVEPOINT statements.

You can create thresholds by using the CREATE THRESHOLD statement. You can alter thresholds by using the ALTER THRESHOLD statement. You can drop thresholds by using the DROP THRESHOLD statement.

You can view your thresholds by querying the SYSCAT.THRESHOLDS view.

Activity and aggregate thresholds

Two types of workload management thresholds are supported: activity thresholds and aggregate 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, the activity threshold is deactivated for that activity. For example, assume that you define a time threshold of 5 minutes and the action for this threshold is CONTINUE. If an activity violates this threshold, the threshold is applied once, not reapplied every 5 minutes.

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 corresponding action is triggered. The work that caused the upper boundary to be violated is the only one affected by the triggered action.

Threshold summary

The table in this topic provides a quick summary of the available thresholds, with their corresponding definition domains and enforcement scopes.

Table 12. Thresholds with definition domains and enforcement scopes

	Thresholds with database enforcement scope	Thresholds with database partition enforcement scope	Thresholds with workload occurrence enforcement scope
Thresholds with database threshold domain	• "CONCURRENTDBCOORDACTIVITIES threshold" on page 65 • "ESTIMATEDSQLCOST threshold" on page 58 • "SQLROWSRETURNED threshold" on page 59 • "ACTIVITYTOTALTIME threshold" on page 60 • "CONNECTIONIDLETIME threshold"	• "TOTALDBPARTITIONCONNECTIONS threshold" on page 61 • "SQLTEMPSPACE threshold" on page 58	N/A
Thresholds with work action set threshold domain	• "CONCURRENTDBCOORDACTIVITIES threshold" on page 65 • "ESTIMATEDSQLCOST threshold" on page 58 • "SQLROWSRETURNED threshold" on page 59 • "ACTIVITYTOTALTIME threshold" on page 60	• "SQLTEMPSPACE threshold" on page 58	N/A
Thresholds with service superclass threshold domain	• "CONCURRENTDBCOORDACTIVITIES threshold" on page 65 • "ESTIMATEDSQLCOST threshold" on page 58 • "SQLROWSRETURNED threshold" on page 59 • "ACTIVITYTOTALTIME threshold" on page 60 • "CONNECTIONIDLETIME threshold"	• "TOTALSCPARTITIONCONNECTIONS threshold" on page 62 • "SQLTEMPSPACE threshold" on page 58	N/A
Thresholds with service subclass threshold domain	• "CONCURRENTDBCOORDACTIVITIES threshold" on page 65 • "ESTIMATEDSQLCOST threshold" on page 58 • "SQLROWSRETURNED threshold" on page 59 • "ACTIVITYTOTALTIME threshold" on page 60	• "SQLTEMPSPACE threshold" on page 58	N/A
Thresholds with workload definition threshold domain	N/A	• "CONCURRENTWORKLOADOCCURRENCES threshold" on page 63	• "CONCURRENTWORKLOADACTIVITIES threshold" on page 63

Activity thresholds

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).

Type Activity

Definition domain
Database or service superclass

Enforcement scope
Database

Tracked work
User connections

Queuing
No

Unit Time duration expressed in minutes, hours, or days

Predictive or reactive
Reactive

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.

This threshold has a granularity of 5 minutes, so all values that you specify for the threshold are rounded to the nearest nonzero multiple of 5 minutes.

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

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

Activities tracked by this threshold are as follows:
- DML activities that are issued at the coordinator partition.
- Nested DML activities that are invoked from a user application. Consequently, DML activities that are issued by the data server internally, such as DML issued from within the DB2 utilities, SYSPROC stored procedures, internal SQL, and so on) 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.

For information about the activities that fall under the work class with the DML work type, see “Work class work types and SQL statements” on page 81.

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.

SQLTEMPSPACE threshold

The SQLTEMPSPACE threshold specifies the maximum amount of temporary table space that can be consumed by a DML activity at any database partition. DML activities often use temporary table space for operations such as sorting and the manipulation of intermediate result sets.

Type Activity

Definition domain
Database, service superclass, service subclass, work action

Enforcement scope
Database partition

Tracked work
See the information later in this topic

Queuing
No

Unit Amount of temporary table space expressed in kilobytes (KB), megabytes (MB), or gigabytes (GB)

Predictive or reactive
Reactive

Activities tracked by this threshold are as follows:
- DML activities that are issued at the coordinator partition.
- 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.

SQLROWSRETURNED threshold

The SQLROWSRETURNED threshold specifies the maximum number of rows that can be returned by the data server to the client.

Type Activity

Definition domain
Database, service superclass, service subclass, work action

Enforcement scope
Database

Tracked work
See the information later in this topic

Queuing
No

Unit Number of rows

Predictive or reactive
Reactive

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 partition.

- Nested DML activities that are invoked from a user application. Consequently, DML activities that are issued by the data server internally, such as DML issued from within the DB2 utilities, SYSPROC stored procedures, internal SQL, and so on) 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.

ACTIVITYTOTALTIME threshold

The ACTIVITYTOTALTIME threshold specifies the maximum amount of time that the data server should spend processing an activity.

Type Activity

Definition domain
Database, service superclass, service subclass, work action

Enforcement scope
Database

Tracked work
Recognized coordinator and nested activities (see "Activities" on page 18)

Queuing
No

Unit Time duration expressed in minutes, hours, or days

Predictive or reactive
Reactive

In situations where the activity is queued by a queuing threshold, the total activity time includes the time spent in the queue awaiting execution. When a cursor is opened, the activity associated with the cursor lasts until the cursor is closed.

This threshold has a granularity of 5 minutes. Therefore, all values that you specify for this threshold are rounded to the nearest nonzero multiple of 5 minutes.

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.

Activity threshold scope resolution

Because activity thresholds apply to individual activities, when multiple thresholds apply to one activity, a decision must be made as to which threshold to enforce. This issue does not exist for aggregate thresholds because the same activity can contribute to multiple aggregates simultaneously (for example, as occurs with concurrency thresholds).

Resolution of the 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. Following is the hierarchy of domains, from local to global:

1. Workload
2. Service subclass
3. Service superclass
4. Work action
5. Database

As an example, a maximum execution time of 1 hour for all database queries defined in the database domain can be overridden by a maximum execution time of 5 hours for the service superclass LARGE QUERIES, which can be overridden by a maximum execution time of 10 hours for the service subclass VERY LARGE QUERIES. Similarly, the maximum execution time of 1 hour defined in the database domain can be overridden by a value of 10 minutes in a different service superclass geared towards important queries that complete quickly.

Aggregate thresholds

TOTALDBPARTITIONCONNECTIONS threshold

The TOTALDBPARTITIONCONNECTIONS threshold specifies the maximum number of concurrent database connections on a coordinator partition for a database. That is, this threshold controls the maximum number of clients that can connect to the database on each of its database partitions. This threshold is not enforced for users with DBADM authority.

Type Aggregate

Definition domain
Database

Enforcement scope
Database partition

Tracked work
Connections

Queuing
Yes (enforced at 0)

Unit Number of concurrent connections

Predictive or reactive
Predictive

For example, if you set the TOTALDBPARTITIONCONNECTIONS threshold to 10 and the database has five partitions, each partition can have up to 10 clients connected concurrently, for a total of 50 client connections across the entire database.

The TOTALDBPARTITIONCONNECTIONS 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 TotalDBPartitionConnections threshold on a database partition ensures that client connections from one database cannot use all of the available connections on a database partition.

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 TOTALDBPARTITIONCONNECTIONS 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.

TOTALSCPARTITIONCONNECTIONS threshold

The TOTALSCPARTITIONCONNECTIONS threshold specifies the maximum number of concurrent database connections on a coordinator partition for a service superclass.

Type Aggregate

Definition domain
Service superclass

Enforcement scope
Database partition

Tracked work
Connections

Queuing
Yes

Unit Number of concurrent connections in service class

Predictive or reactive
Predictive

When the TOTALSCPARTITIONCONNECTIONS 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 TOTALSCPARTITIONCONNECTIONS threshold, the connection is considered to be in a *transient* state.

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 and returns the ticket that is held by the TOTALSCPARTITIONCONNECTIONS threshold. 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 TOTALSCPARTITIONCONNECTIONS threshold controls only coordinator connections. Connections made by subagents are not counted towards the threshold.

If you set a threshold value for TOTALDBPARTITIONCONNECTIONS, set it large enough to accommodate the threshold that you specify for TOTALSCPARTITIONCONNECTIONS. For example, if you define five service superclasses for a database and each of them has a TOTALSCPARTITIONCONNECTIONS threshold value of 10, the TOTALDBPARTITIONCONNECTIONS threshold value should be at least 50.

CONCURRENTWORKLOADOCCURRENCES threshold

The CONCURRENTWORKLOADOCCURRENCES threshold is an aggregate threshold that specifies the maximum number of workload occurrences that can run concurrently on the coordinator partition.

Type Aggregate

Definition domain
Workload

Enforcement scope
Database partition

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 partitions, the work on these partitions does not count towards the concurrent threshold total on the coordinator partition. For example, assume that a CONCURRENTWORKLOADOCCURRENCES threshold is defined to permit only one occurrence of workload A on a database partition. Then assume that an application connects to database partition 1, resulting in an occurrence of workload A being started, and that this workload causes work to be sent to database partitions 1, 2, and 3. In this situation, the total number of occurrences of workload A is one on database partition 1 and zero on database partitions 2 and 3. Therefore, if another application connects to database partition 1 and another occurrence of workload A is started on database partition 1, that workload is rejected. However, new occurrences of workload A can still be started on database partitions 2 and 3.

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 18)

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 included all recognized coordinator activities and any nested activities that are generated as a result of the execution of the coordinator activity. For example, 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 class work types and SQL statements” on page 81 are not counted. Similarly, nested subagent activities such as RPC requests, DSS requests, and nested DSS requests are not counted either.
- 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 workload activities is 4.
   - The BOB.SP1 stored procedure calls the SYSPROC.SP2 stored procedure. The current number of workload activities is 5.
     – The SYSPROC.SP2 stored procedure issues some SQL. The current number of workload activities is 5.
     – The SYSPROC.SP2 stored procedure ends. The current number of workload activities is 4.
   - The BOB.SP1 stored procedure calls the BOB.SP2 stored procedure. The current number of workload 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 workload activities is 4.
   - The BOB.SP1 stored procedure ends. The current number of workload activities is 0.
7. Open a cursor C2. The current number of workload activities is 1.
8. CALL the BOB.SP2 stored procedure. The current number of workload activities is 2.

CONCURRENTDBCOORDACTIVITIES threshold

The CONCURRENTDBCOORDACTIVITIES threshold specifies the maximum number of recognized coordinator activities that can run concurrently across all database partitions in the specified definition domain. If an application starts more than one concurrent activity, it might have to pass this threshold more than once, potentially consuming the concurrency available for this threshold and creating a *self-deadlock* scenario.

Type Aggregate

Definition domain
Database, work action, service superclass, service subclass

Enforcement scope
Database

Tracked work
Recognized coordinator and nested activities (see “Work class work types and SQL statements” on page 81)

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 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. Similar to the CONCURRENTWORKLOADACTIVITIES threshold, the CONCURRENTDBCOORDACTIVITIES threshold tracks coordinator activities and any nested activities generated. Unlike the CONCURRENTWORKLOADACTIVITIES threshold, the CONCURRENTDBCOORDACTIVITIES threshold is a queuing threshold.

When creating queuing thresholds of the CONCURRENTDBCOORDACTIVITIES type, be aware of configurations that might lead to 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 (or calls a stored procedure) that the DB2 data server recognizes as activity A1, which consumes the unique ticket that is available for the threshold.
3. While the activity A1 is still active, the application now issues a second SQL 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 situation 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.

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. You can also use time thresholds to prevent an activity from remaining queued indefinitely, which allows scenarios like this one to resolve themselves without intervention.

Threshold evaluation order

Although certain thresholds are evaluated independently of other thresholds (because they are driven by specific events such as a new connection or a new workload occurrence), other thresholds are interdependent and must be evaluated in a specific order when you define them in the same database.

The evaluation of thresholds occurs as follows:

- CONCURRENTWORKLOADOCCURRENCES. This threshold is evaluated independently when a new workload occurrence is started for a workload definition that has this threshold applied to it.
- TOTALDBPARTITIONCONNECTIONS. This threshold is evaluated independently when a new connection is made to a database.
- TOTALSCPARTITIONCONNECTIONS. 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).

The remaining thresholds are all based on recognized activities resulting from an SQL statement or the execution of a utility such as the load utility. Predictive thresholds are checked before reactive thresholds because a check must be done to ensure that predictive thresholds are not violated before a database activity can start to run. The order in which predictive thresholds are evaluated is as follows.

Note: If you do not define a threshold, its step is skipped. Also, the steps described might be combined at run time for performance reasons.

1. 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 "Activity threshold scope resolution" on page 60 for more information). The result of this operation is one value of ESTIMATEDSQLCOST applicable to the activity.
2. Check if an ESTIMATEDSQLCOST 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.
3. 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.
4. Check if a CONCURRENTDBCOORDACTIVITIES 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.
5. Check if a CONCURRENTDBCOORDACTIVITIES 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.
6. Check if a CONCURRENTDBCOORDACTIVITIES 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.
7. Check if a CONCURRENTDBCOORDACTIVITIES 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.

The evaluation order for concurrency thresholds does not follow the hierarchy used for resolving activity thresholds (see "Activity threshold scope resolution" on page 60 for more information). The database-level work action set concurrency thresholds are checked first to avoid the following situation. Assume that the following thresholds are defined:

- 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 when a new LOAD activity enters as the 10th activity. If the scope resolution hierarchy is used for the 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 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 10th LOAD activity is then queued.

The result of this situation is that 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 not desirable 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 work action threshold condition). Therefore, to avoid this type of situation, the work action concurrency threshold is checked first. Because the concurrency threshold is checked first, the 10th activity in the service class (which happens to be a LOAD activity) is blocked at the work action threshold level before it can attempt to consume one spot in the service superclass S1.

Reactive threshold considerations

Reactive thresholds are evaluated in a discrete fashion when an activity is executing. No specific order is used to evaluate the reactive threshold SQLTEMPSPACE, SQLROWSRETURNED, or ACTIVITYTOTALTIME.

Working with thresholds

Creating a threshold

Create thresholds using the DDL statement CREATE THRESHOLD. You create a threshold to impose a limit on resource consumption.

To create a threshold, you require DBADM or SYSADM authority.

See the following topics for more information about prerequisites:
- Appendix A, “Workload management DDL statement considerations,” on page 267
- 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.

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.

     Note: The domain that applies depends on the type of threshold. See “Threshold summary” on page 57 for more information.
     The domain can be one of the following:
     – Database
     – Service superclass
     – Service subclass
     – Workload

     For more information, see Chapter 4, “Thresholds,” on page 55.
   - The enforcement scope for the threshold. The threshold scope is the enforcement range of the threshold in its domain .

Note: The enforcement scope that applies depends on the type of threshold. See "Threshold summary" on page 57 for more information.
The enforcement scope can be one of the following:
– Database
– Database partition
– Workload occurrence

For more information, see Chapter 4, "Thresholds," on page 55.

- Optional: A value that specifies whether the threshold is enabled or disabled. By default, the threshold is created as enabled. If you create the threshold as disabled and want to enable it later, use the ALTER THRESHOLD statement. For more information, see "Altering a threshold" on page 73.

  Note: If you have work action thresholds, enable and disable them using the ALTER WORK ACTION SET statement.
- The threshold predicate to specify the maximum value for the threshold. When the maximum value is violated, the action specified for the threshold is enforced . The thresholds are as follows:
  – TOTALDBPARTITIONCONNECTIONS. Specify `0` or a positive integer to specify the maximum number of concurrent coordinator connections that can run on a database partition. A value of `0` prevents new coordinator connections from connecting. The definition domain must be DATABASE, and its enforcement scope must be PARTITION. For example, to indicate that five concurrent coordinator connections can run on a database partition:

    ```
    TOTALDBPARTITIONCONNECTIONS > 5
    ```

    Note: This threshold does not apply to users with DBADM authority.
  – TOTALSCPARTITIONCONNECTIONS. Specify `0` or a positive integer to specify the maximum number of concurrent coordinator connections that can run on a database partition in a service superclass. A value of `0` prevents new connections from joining the service superclass. The definition domain must be SERVICE SUPERCLASS and the enforcement scope must be PARTITION. Because this threshold is a queueing threshold, it takes an optional AND QUEUEDCONNECTIONS keyword. Valid arguments are as follows:
    - AND QUEUEDCONNECTIONS > `0` indicates that coordinator connections are not queued.
    - AND QUEUEDCONNECTIONS > *integer* indicates the maximum queue size when the number of coordinator connections is exceeded.
    - AND QUEUEDCONNECTIONS UNBOUNDED indicates that there is no upper limit to the size of the queue. In this situation, the threshold violated condition is never met.

    For example, to indicate that five concurrent coordinator connections can run on a database partition in a service superclass, and that seven connections can be queued:

    ```
    TOTALSCPARTITIONCONNECTSIONS > 5 AND QUEUEDCONNECTIONS > 7
    ```

    Note: You can use TOTALSCPARTITIONCONNECTIONS to effectively disable service classes that cannot be manually disabled (for example, the default user class). Although thresholds do not apply to users with DBADM authority running in the SYSDEFAULTADMWORKLOAD service class, a disabled service class is not available to any user.

– CONNECTIONIDLETIME. Specify a positive integer and one of the following keywords to specify the maximum amount of time that a connection can remain idle:
  - DAY
  - DAYS
  - HOUR
  - HOURS
  - MINUTE
  - MINUTES

  Because the minimum granularity is 5 MINUTES, all values that you specify are rounded to nearest nonzero multiple of 5 MINUTES. The definition domain must be DATABASE or SERVICE SUPERCLASS, and the enforcement scope must be DATABASE. For example, to indicate that a connection can be idle for 90 minutes, specify:

  ```
  CONNECTIONIDLETIME > 90 MINUTES
  ```

  The maximum value that can be specified for this threshold is 2147483400 seconds. Any value specified (using the DAY, HOUR, MINUTE or SECOND keyword) that is larger than 2147483400 seconds is truncated to the maximum value.
– CONCURRENTWORKLOADOCCURRENCES. Specify a nonzero positive integer to specify the maximum number of concurrent workload occurrences on a database partition. The definition domain must be WORKLOAD. For example, to indicate that a maximum of eight occurrences of a workload can concurrently occur on a database partition:

  ```
  CONCURRENTWORKLOADOCCURRENCES > 8
  ```
– CONCURRENTWORKLOADACTIVITIES. Specify a nonzero positive integer to specify the maximum number of concurrent coordinator and nested activities on a database partition for a workload occurrence. The enforcement scope must be WORKLOAD OCCURRENCE. For example, to indicate that a maximum of 26 activities can execute concurrently on a database partition for a workload occurrence:

  ```
  CONCURRENTWORKLOADACTIVITIES > 26
  ```
– CONCURRENTDBCOORDACTIVITIES. Specify `0` or a positive integer to specify the maximum number of concurrent coordinator activities that can run on the all database partitions in the specified domain. A value of `0` prevents new coordinator activities (including any activity to disable or alter this threshold) from executing. In this situation, you must use the SYSDEFAULTADMWORKLOAD workload to disable or alter the threshold. The definition domain can be DATABASE, SERVICE SUPERCLASS, or SERVICE SUBCLASS. The enforcement scope must be DATABASE. Because this threshold is a queueing threshold, it takes an optional AND QUEUEDACTIVITIES keyword. Valid arguments are as follows:
  - AND QUEUEDACTIVITIES > 0 indicates that coordinator activities are not queued.
  - AND QUEUEDACTIVITIES > *integer* indicates the maximum queue size when the number of coordinator activities is exceeded.
  - AND QUEUEDACTIVITIES UNBOUNDED indicates that there is no upper limit to the size of the queue. In this situation, the threshold violated condition is never met.

For example, to indicate that 12 concurrent coordinator activities can run in the specified definition domain and that 9 activities can be queued:

```
CONCURRENTDBCOORDACTIVITIES > 12 AND QUEUEDACTIVITIES > 9
```

– ESTIMATEDSQLCOST. Specify a nonzero positive big integer to specify the maximum optimizer-assigned cost of a coordinator DML activity, or a DML activity that is invoked by user logic. The definition domain can be DATABASE, SERVICE SUPERCLASS, or SERVICE SUBCLASS. The enforcement scope must be DATABASE. For example, to indicate that no DML activity larger than 1 000 timerons can run in the database, specify:

  ```
  ESTIMATEDSQLCOST > 1000
  ```

– SQLROWSRETURNED. Specify a nonzero positive integer to specify the maximum number of rows that can be returned to the client application from the application server. The definition domain can be DATABASE, SERVICE SUPERCLASS, or SERVICE SUBCLASS. The enforcement scope must be DATABASE. For example, to indicate that no more than 50 000 rows can be returned, specify:

  ```
  SQLROWSRETURNED > 50000
  ```

– ACTIVITYTOTALTIME. Specify a nonzero positive integer to specify the maximum amount of time that the activity can execute, including the amount of time that the activity can be queued. This value is followed by one of the following duration keywords:
  - DAY
  - DAYS
  - HOUR
  - HOURS
  - MINUTE
  - MINUTES

  Because the minimum granularity is 5 MINUTES, all values specified are rounded to nearest nonzero multiple of 5 MINUTES. The definition domain can be DATABASE, SERVICE SUPERCLASS, or SERVICE SUBCLASS. The enforcement scope must be DATABASE. For example, to specify that the maximum activity time can be two hours, specify:

  ```
  ACTIVITYTOTALTIME > 2 HOURS
  ```

  The maximum value that can be specified for this threshold is 2147483400 seconds. Any value specified (using the DAY, HOUR, MINUTE or SECOND keyword) that is larger than 2147483400 seconds is truncated to the maximum value.

– SQLTEMPSPACE. Specify a nonzero positive integer to specify the maximum amount of temporary table space that can be consumed at a database partition. This value is followed by one of the following space keywords:
  - K (kilobytes)
  - M (megabytes)
  - G (gigabytes)

  The definition domain can be DATABASE, SERVICE SUPERCLASS, or SERVICE SUBCLASS. The enforcement scope must be DATABASE PARTITION. For example, to specify that the maximum amount of temporary table space that can be used on a database partition is 100 MB:

  ```
  SQLTEMPSPACE > 100 M
  ```

- The actions to be taken if the threshold boundary is violated. The actions consist of a mandatory progress action and an optional collect activity action. The collect activity action specifies which information should be collected for the activity that caused the threshold boundary to be violated. Specify an action only for activity-related thresholds; an action is ignored if you specify it for non-activity-related thresholds.
  - STOP EXECUTION. For the TOTALDBPARTITIONCONNECTIONS and TOTALSCPARTITIONCONNECTIONS thresholds, the connection is prevented from being established. For CONNECTIONIDLETIME thresholds, the connection is closed. For CONCURRENTWORKLOADOCCURRENCES, the new workload occurrence is prevented from being created. For all other activity-related thresholds, the activity that causes the threshold to be violated is stopped. If a THRESHOLDVIOLATIONS event monitor is active, a record is written to the event monitor indicating that the threshold was violated.
  - CONTINUE. If a THRESHOLDVIOLATIONS event monitor is active, a record is written to the event monitor indicating that the threshold was violated. Otherwise no further action is taken.

    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.
  - COLLECT ACTIVITY DATA. The information to collect for the activity event monitor. The default is COLLECT ACTIVITY DATA NONE.
    - NONE. No activity information is collected when the activity that violated the threshold completes execution.
    - The location where data is to be collected:
      - ON COORDINATOR DATABASE PARTITION. Activity data is only collected at the coordinator partition for the activity if an ACTIVITIES event monitor is active on that database partition.
      - ON ALL DATABASE PARTITIONS. For predictive thresholds whose action is CONTINUE only, activity data is collected at all database partitions where the activity is processed, however activity details or values are only collected at the coordinator partition. For reactive thresholds and for any threshold whose action is STOP EXECUTING, this option has the same effect as ON COORDINATOR DATABASE PARTITION. Activity data is only collected on those database partitions that have an active ACTIVITIES event monitor.
    - WITHOUT DETAILS. Information about each activity that violates the threshold is sent to the applicable event monitor when the activity completes execution. Statement and compilation environment however, is not sent to the event monitor.
    - WITH DETAILS. Statement and compilation environment information is sent to the applicable event monitor for those activities that have them. When you request details, you can also specify AND VALUES to have input data values sent to the applicable event monitor for those activities that have them.

  For example, to send all available information about the activity that violated a threshold condition (including the statement that caused the threshold to be violated, the compilation environment and input data values) to the activity event monitor when the activity completes, use the following keywords in the CREATE THRESHOLD statement:

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

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 DDL statement ALTER THRESHOLD. You could alter a threshold if it does not produce the results that you expect.

To alter a threshold, you require DBADM or SYSADM authority.

See Appendix A, “Workload management DDL statement considerations,” on page 267 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. For more information, see ALTER WORK ACTION SET statement.

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. See “Creating a threshold” on page 68 for an explanation of the supported values for these properties.
   - The new boundary for the threshold predicate.

     Note: You cannot alter the threshold type (that is, you cannot change a TOTALDBPARTITIONCONNECTIONS threshold to a TOTALSCPARTITIONCONNECTIONS threshold).
   - The actions to be taken if the threshold boundary is violated.
   - Whether the threshold is enabled or disabled. Activities run under the applicable thresholds that were enabled when that activity started.
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.

To drop a threshold, you require DBADM or SYSADM authority.

See Appendix A, “Workload management DDL statement considerations,” on page 267 for more information about prerequisites.

Note: If you want to drop a threshold in a work action set, use the ALTER WORK ACTION SET statement.

To drop a threshold:

1. If the threshold is a queuing threshold, use the ALTER THRESHOLD statement to disable it.
2. If you disabled a queuing threshold by using an ALTER THRESHOLD statement, issue a COMMIT statement to commit the change.
3. Use the DROP THRESHOLD statement to drop the threshold.
4. Commit your changes. When you commit your changes the threshold is removed from the SYSCAT.THRESHOLDS view.

Chapter 5. Work action sets, work actions, work class sets, and work classes

Using work action sets, work actions, work class sets, and work classes, you can classify activities based on *what* they are in a similar fashion to how you can use workloads to classify activities based on *who* submitted them. When you classify activities based on attributes such as their activity type, you can treat these activities differently by applying different types of actions to them.

You can classify activities into different work classes and group work classes into work class sets. Whey you apply a work action set, which can contain work actions, to a work class set, the work actions in the work action set can be applied to the work classes in the work class set. Because the objects that classify the activities (work classes and work class sets) are separate from the objects that define the actions to be applied to the activities (work actions and work action sets), the classification objects can be shared among more than one work action set and work action.

Work classes and work class sets

A work class is a method of categorizing individual database activities based on attributes of the database activity. 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 include: activity type (DDL, DML, LOAD), the estimated cost (where available), the estimated cardinality (where available), and the schema (where available).

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, READ, WRITE, DML, DDL, and LOAD), 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 class work types and SQL statements" on page 81.
  - 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.

    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 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 83.
- 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.

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.

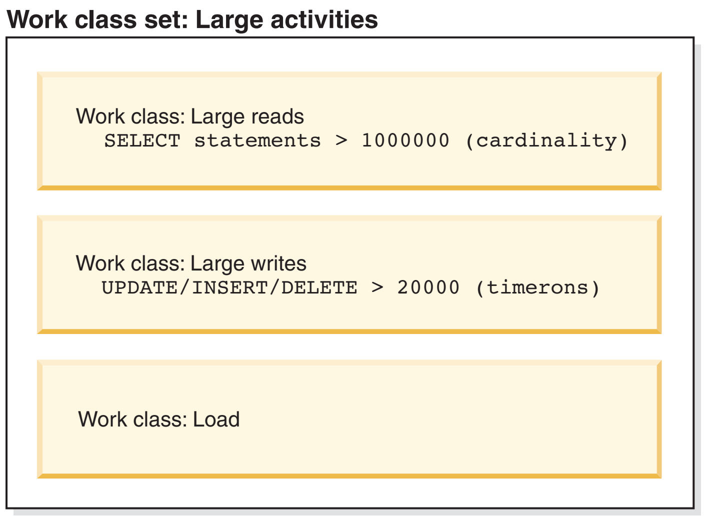

Work class set: Large activities
Work class: Large reads
SELECT statements > 1000000 (cardinality)
Work class: Large writes
UPDATE/INSERT/DELETE > 20000 (timerons)
Work class: Load

Figure 14. 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 either a service superclass or the 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.

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.

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.

  If no work actions are applied to a work class, the data server behaves as though the work class does not exist. 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 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 or a service superclass.

  When a work action set is applied to a database (depending on the work classes that the work actions are associated with), the work action set applies to some or all activities that enter the database. When a work action set is applied to a service superclass (depending on the work classes that the work actions are associated with), that work action set applies to some or all activities that are run under that service superclass. For example:
  - A work action set that is applied to a database 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 83.
- 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 microseconds 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 117.
- 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.

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

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 the work class 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 object that the work action set is associated with (database or service superclass).
- 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 or a service superclass. You must also specify which work class set the work action set is to work with. This allows 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 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.

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.

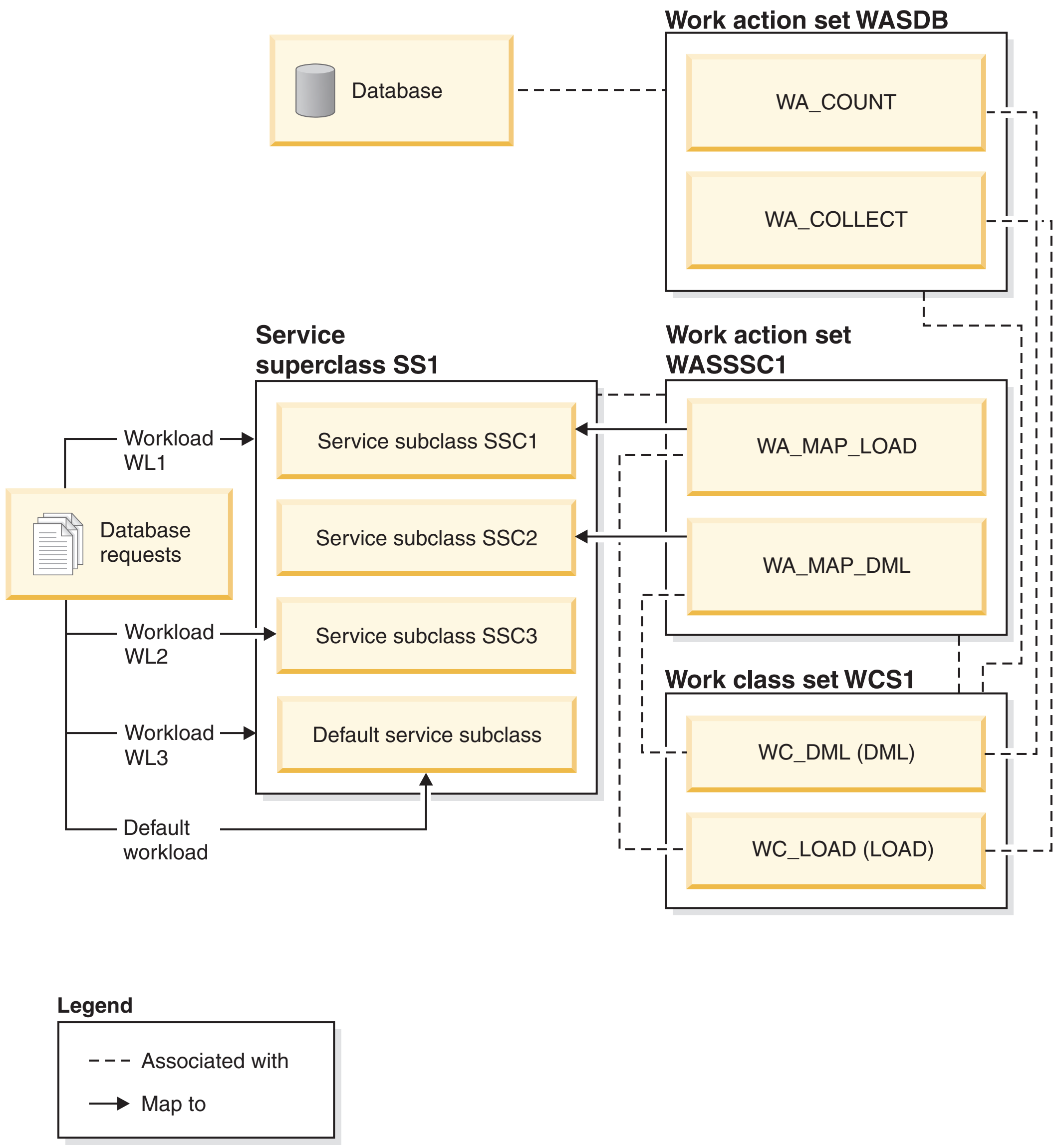


Figure 15. Overview of work action sets and work class sets

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, any activities that are assigned to the default user workload, the WL1 workload, or the WL3 workload and fall under the WC_DML or WC_LOAD work classes 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 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. 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 either the WL2 workload or to the default workload (because their connection attributes do not match any defined workload) are mapped directly to service subclasses. Specifically, the WL2 workload maps its activities to the SSC3 service subclass. When a workload maps activities directly to a service subclass, no work actions are applied to those activities.

Work class work types and SQL statements

A work class has an associated work type. One or more activity types can fall under a work class type. Before defining a work class, you should understand what types of activities fall under each type of work class.

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 command, all the statements in the table below 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 13. Work type keywords and associated SQL statements

Work type keyword	Applicable SQL statements
READ	• All SELECT statements (select into, values into, full select) **Note:** SELECT statements containing a DELETE, INSERT, or UPDATE are not included • All XQuery statements
WRITE	• 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. **Note:** The CALL statement is only classified under the CALL and ALL work class types.
DML	All statements that are classified under the READ and WRITE work class types

Table 13. Work type keywords and associated SQL statements (continued)

Work type keyword	**Applicable SQL statements**
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 **Note:** The load utility is only classified under the LOAD and ALL work class types.
ALL	All database activity. **Note:** 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 152.

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

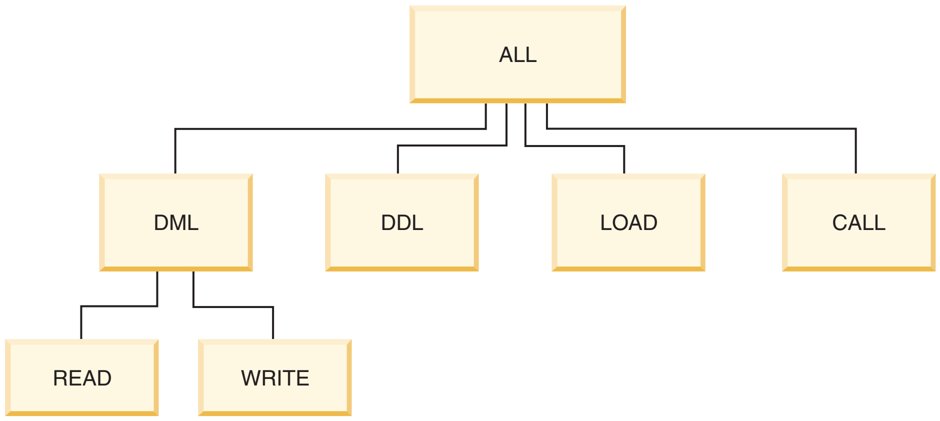


Figure 16. Work type keywords

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.

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. Only work classes with work actions applied to them are considered.

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.

Work actions and the work action set domain

You can define a work action set for either a database or a service superclass. 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 one of the following actions:

- A threshold

  The actual threshold is specified by the `WHEN threshold-type` keyword. 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 allowed 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. See "Collecting data for individual activities" on page 126 for more information.
- 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. See "Collecting data for individual activities" on page 126 for more information.

If the work actions in the work action set 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 one 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

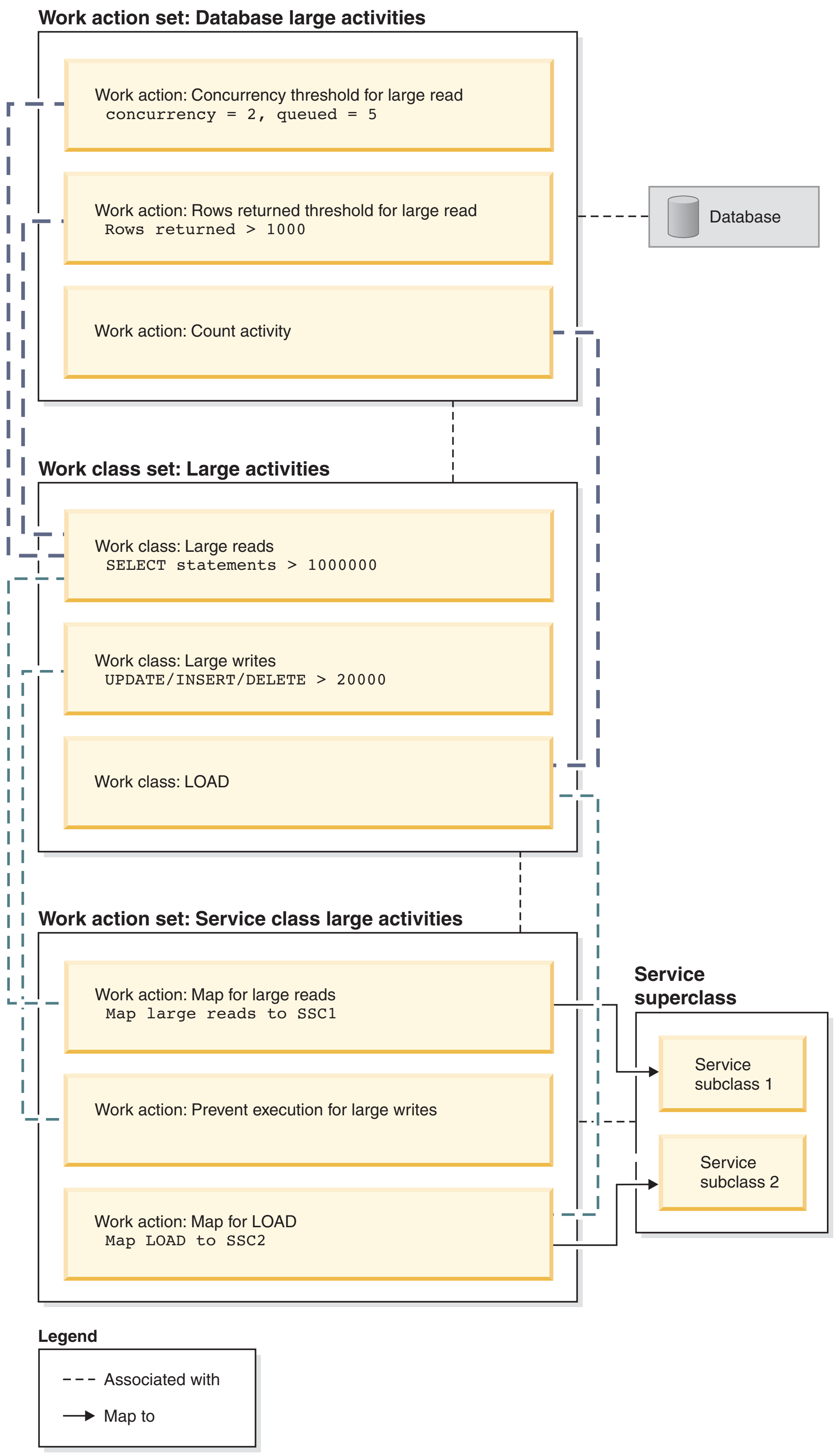
  Behavior is the same as for the database work action.
- COLLECT AGGREGATE ACTIVITY DATA

  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 are not any of these actions, SQL4720N is returned.

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.



Work action set: Database large activities
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
Database
Work class set: Large activities
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
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
Service superclass
Service subclass 1
Service subclass 2
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 allows 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 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 are defined for databases can contain work actions that specify thresholds.

The following thresholds are supported:
- Aggregate threshold:
  - CONCURRENTDBCOORDACTIVITIES
- Activity thresholds:
  - SQLTEMPSPACE
  - SQLROWSRETURNED
  - ACTIVITYTOTALTIME
  - ESTIMATEDSQLCOST

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 table shows which work class categories are supported by which threshold types:

Table 14. Work classification supported by thresholds

	"CONCURRENTDBCOORDACTIVITIES threshold" on page 65	"SQLTEMPSPACE threshold" on page 58	"SQLROWSRETURNED threshold" on page 59	"ESTIMATEDSQLCOST threshold" on page 58	"ACTIVITYTOTALTIME threshold" on page 60
READ	Yes	Yes	Yes	Yes	Yes
WRITE	Yes	Yes	Yes	Yes	Yes
CALL	Yes	No	No (see note)	No	Yes

Table 14. Work classification supported by thresholds (continued)

	"CONCURRENTDBCOORDACTIVITIES threshold" on page 65	"SQLTEMPSPACE threshold" on page 58	"SQLROWSRETURNED threshold" on page 59	"ESTIMATEDSQLCOST threshold" on page 58	"ACTIVITYTOTALTIME threshold" on page 60
DML	Yes	Yes	Yes	Yes	Yes
DDL	Yes	No	No	No	Yes
LOAD	Yes	No	No	No	Yes
ALL	Yes	Some	Some	Some	Yes

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.

Assignment of activities to work classes

If a work class set, through a work action set, is associated with either a database or a service superclass, 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, and all of the work classes have a work action applied to them:

- 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 152.

If a work class does not have a work action applied to it, that work class is ignored, and no activities are assigned to it. For more information, see "Evaluation order of work classes in a work class set" on page 83.

Application of work actions to database activities

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

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.

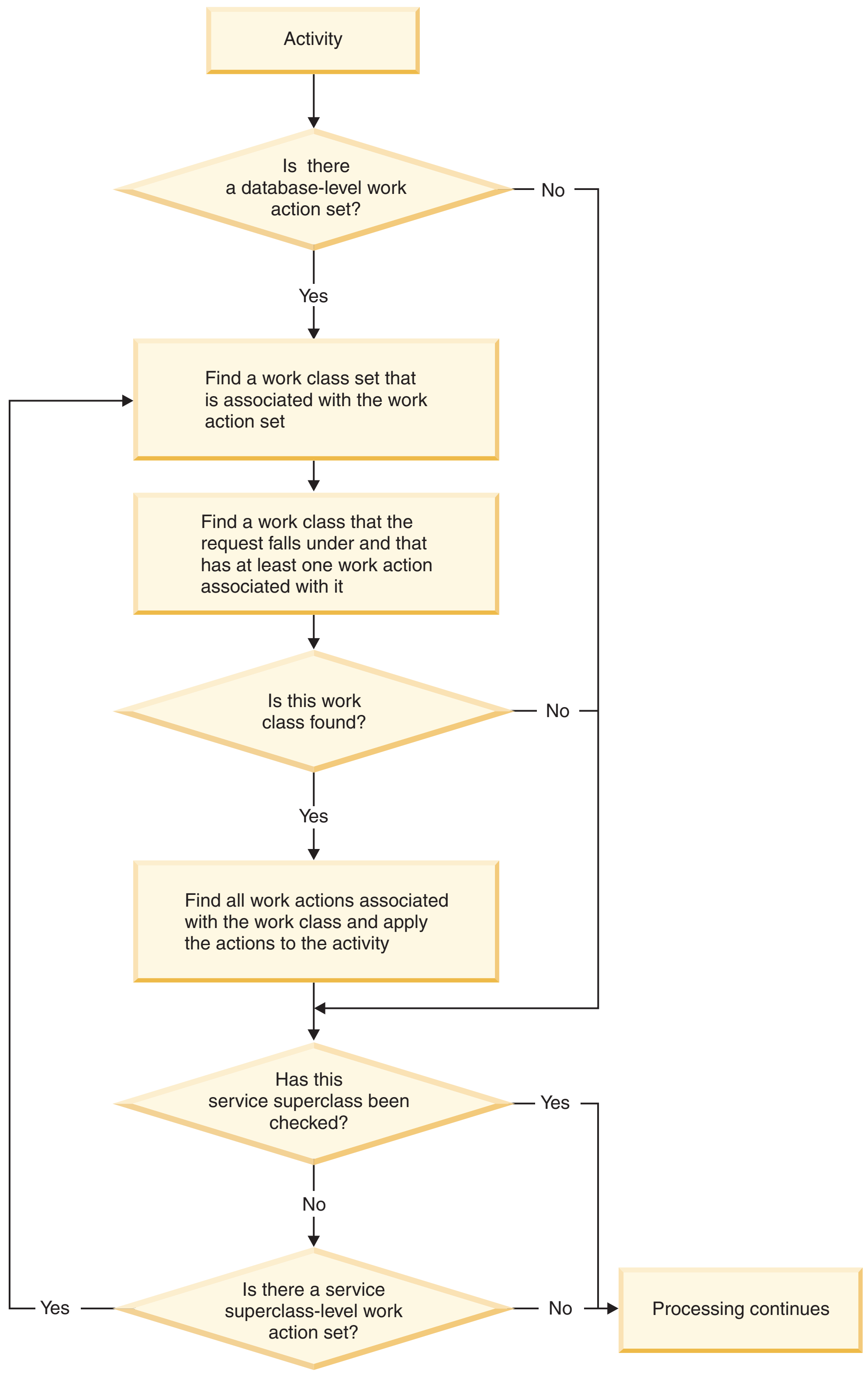

Activity
Is there a database-level work action set?
No
Yes
Find a work class set that is associated with the work action set
Find a work class that the request falls under and that has at least one work action associated with it
Is this work class found?
No
Yes
Find all work actions associated with the work class and apply the actions to the activity
Has this service superclass been checked?
Yes
No
Yes
Is there a service superclass-level work action set?
No
Processing continues

Figure 18. Application of a work action 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 work class that has one or more work actions applied to it, those work actions are applied to the activity.
4. 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.
5. 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.
6. If the activity falls under a work class that has one or more work actions applied to it, those work actions are applied to the 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.

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 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.

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 one work action to map a specific type of activity to a service subclass and use a different work action to apply a control known as a threshold to ensure that 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 or to activities in a service superclass (but not both). 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.

The following figure shows an example of a workload management implementation that uses workloads and work action sets. In this figure, assume that a request is assigned to workload WL_A based on the user ID 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.

For example, 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, and imposes a threshold that states that no more than 500 activities can execute concurrently for the entire database. Assuming that the request does not exceed the boundaries established by this threshold, 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.

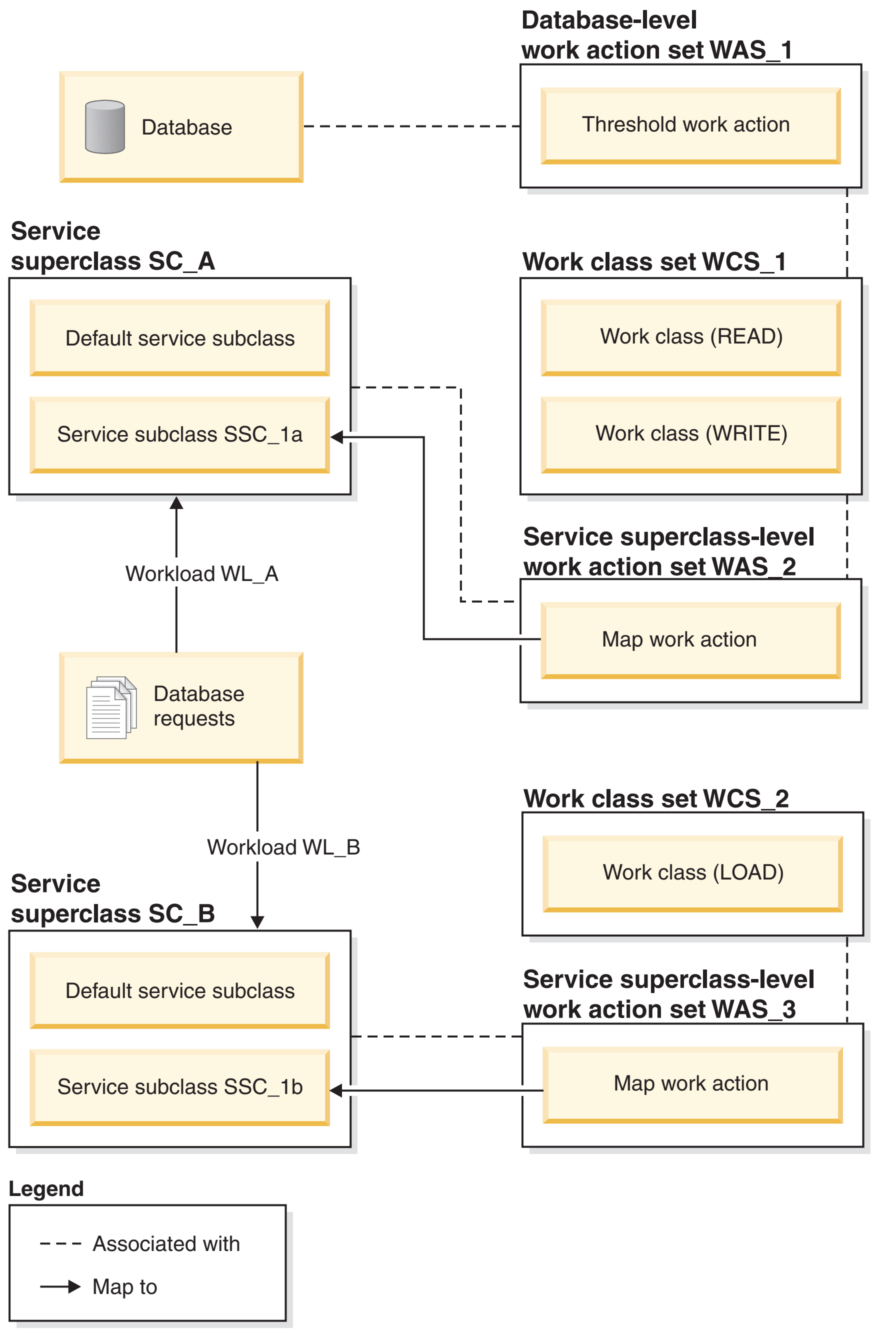


Figure 19. Workloads and work action sets

Working with work action sets and work actions

Creating a work action set

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

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

For additional prerequisites, see the following topics:

- Appendix A, “Workload management DDL statement considerations,” on page 267
- Naming rules

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 the database 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 system service class (SYSDEFAULTSYSTEMCLASS) or the maintenance service class (SYSDEFAULTMAINTENANCECLASS).

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 or 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. 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
     ```
   - 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 94.
   - 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.

To alter a work action set, you require SYSADM or DBADM authority.

For additional prerequisites, see the following topics:

- Appendix A, “Workload management DDL statement considerations,” on page 267
- Naming rules

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 or service superclass) the work action set is defined for. If you want to change the which object the work action set is associated with, you must drop and recreate the work action set.

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 94
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 97.
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 99.
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.
5. Commit your changes. When you commit your changes, the work action set is updated in the SYSCAT.WORKACTIONSETS view. The SYSCAT.WORKACTIONS views 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.

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

At runtime, a disabled work action set is treated as if it does not exist. 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 statements to the service subclass SMALLREADSERVICECLASS. If the READACTIVITIES work action set is disabled, all SELECT statements are treated as though the READACTIVITIES work action set does not exist, and are mapped to the default service subclass.

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:
   - Use the CREATE WORK ACTION SET statement to disable the work action set. For example:
     ```
     CREATE WORK ACTION SET work-action-set-name ... DISABLE
     ```
   - Use the ALTER WORK ACTION SET statement. For example:
     ```
     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.

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

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.

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.

Creating a work action

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

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

For additional prerequisites, see the following topics:
- Appendix A, "Workload management DDL statement considerations," on page 267
- Naming rules

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 the database. For the list of supported thresholds for work actions, see "Thresholds that can be used in work actions" on page 86.
- 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.

- Only one work action of the same type can be applied to the same work class from the same work action set. Thresholds are the exception. You can apply more than one threshold to a 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.

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 `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.
   - If the work action set is associated with a database, 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:
     - CONCURRENTDBCOORDACTIVITIES and its QUEUEDACTIVITIES keyword.
     - SQLTEMPSPACE
     - SQLROWSRETURNED

  - ESTIMATEDSQLCOST
  - ACTIVITYTOTALTIME

    Note: The maximum value that can be specified for the ACTIVITYTOTALTIME threshold is 2147483400 seconds. Any value specified (using the DAY, HOUR, MINUTE or SECOND keyword) that is larger than 2147483400 seconds is truncated to the maximum value.

  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 activity 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 partition, a specific database partition, or from all database partitions. 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 allowed 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 ESTIMATEDSQLCOST > 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 activity 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 partition or from all database partitions. 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 activity 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
  ```

- To collect aggregate activity data for activities that fall under the work class, specify the COLLECT AGGREGATE ACTIVITY DATA keyword. 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 on histograms and histogram templates, see "Histograms in workload management" on page 117.
- 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 actions 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 only takes effect in the database 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.

To alter a work action, you require SYSADM or DBADM authority.

See Appendix A, "Workload management DDL statement considerations," on page 267 for additional prerequisites.

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 on histograms and histogram templates, see “Histograms in workload management” on page 117.
- 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.

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

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:

```
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:

  ```
  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.

- To drop a work action, you require SYSADM or DBADM authority.
- See Appendix A, "Workload management DDL statement considerations," on page 267 for additional prerequisites.

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, 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.

Working with work class sets and work classes

Creating a work class set

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

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

For additional prerequisites, see the following topics:

- Appendix A, "Workload management DDL statement considerations," on page 267
- Naming rules

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 100.
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.

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

For additional prerequisites, see the following topics:

- Appendix A, "Workload management DDL statement considerations," on page 267
- Naming rules

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."
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 103.
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 103. 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.

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

You can only drop a work class set 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.

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.

Creating a work class

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

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

For additional prerequisites, see the following topics:

- Appendix A, "Workload management DDL statement considerations," on page 267
- Naming rules

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 argument. Use this 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:

         ```
         WORK TYPE READ FOR TIMERONCOST FROM 5000 TO UNBOUNDED
         ```
       - WRITE, which represents SQL activities that update data in the database.

         For example, to associate data writing activities that update between 50 and 100 rows with this work class, you would specify:

         ```
         WORK TYPE WRITE 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 a cost in timerons from 500 to 1000 with this work class, you would specify:

         ```
         WORK TYPE DML FOR TIMERONCOST FROM 500 TO 1000
         ```
       - 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:

  ```
  WORK TYPE ALL FOR CARDINALITY FROM 300 TO 1500 ROUTINES
  IN SCHEMA NEWHIRES
  ```

  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.

– 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.

Altering a work class

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

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

See Appendix A, "Workload management DDL statement considerations," on page 267 for additional prerequisites.

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 100 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 SCHEMA keyword 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 POSITION keyword. 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.

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

See Appendix A, "Workload management DDL statement considerations," on page 267 for additional prerequisites.

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.

Part 3. Monitoring and control

Chapter 6. Monitoring and control

With workload management features and functionality, you can both monitor and control the work being executed in the database.

For example, you can perform the following tasks:

- Analyze the workload on your system to help design your initial workload management configuration.
- Track and investigate the behavior of your system by obtaining the following types of operational information:
  - General monitoring information about the environment
  - Information for analyzing system performance degradation
  - Information for diagnosing hung activities
  - Information for investigating agent contention
  - Information for isolating poorly performing queries

  Information is available for activities, service classes, workloads, work classes, threshold queues, and threshold violations.
- Control the environment by canceling queued activities that you expect will cause problems or cancel running activities that you have diagnosed as negatively impacting the system.

With the workload management solution, you can troubleshoot more effectively because you can drill down from the database into service classes and workloads and into individual activities in the database.

Monitoring data overview

Monitoring data is available from workloads, work classes, service subclasses, service superclasses, and threshold queues. You can use this data to diagnose and correct problems and for performance tuning.

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. You can access workload statistics and information about workload occurrences in real time using table functions.

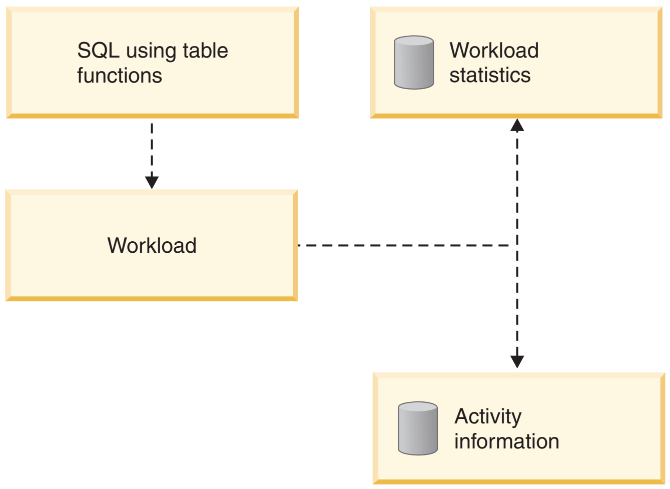


Figure 20. Monitoring data that is available for workloads

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.

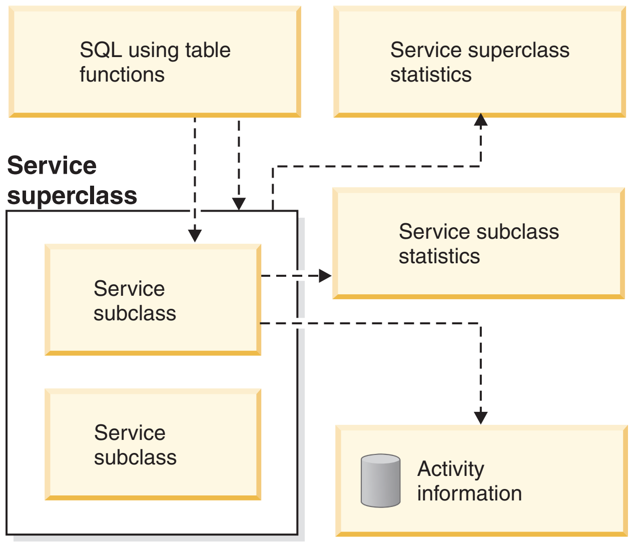

SQL using table functions
Service superclass statistics
Service superclass
Service subclass statistics
Service subclass
Service subclass
Activity information

Figure 21. Monitoring data that is available for service classes

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.

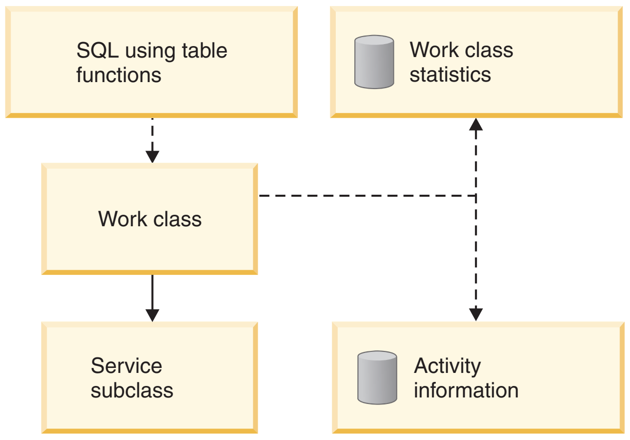


Figure 22. Monitoring data that is available for work classes

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 queueing thresholds). You can access queueing threshold statistics in real time using table functions.

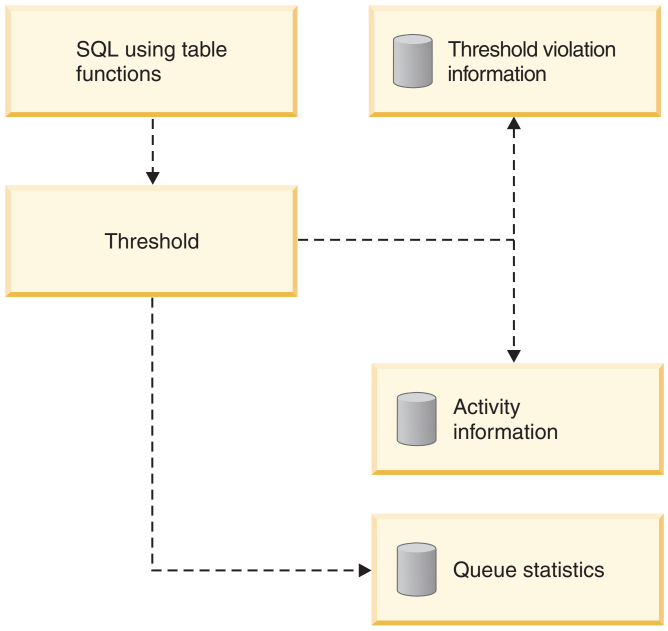


Figure 23. Monitoring data that is available for thresholds

Workload management table functions to obtain operational information

You can use the table functions described in this topic to obtain operational information.

The workload management table functions are available in the SYSPROC schema. They are high performance and can return information about the work occurring in the system with very little impact on currently executing workloads.

You can use the following table functions to examine the work occurring on the system in terms of the service classes, workload occurrences, agents, requests, and activities. All table functions can return information for either a single database partition or for all database partitions in a partitioned database environment.

- WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES (*service_superclass_name*, *service_subclass_name*, *dbpartitionnum*). Use this table function to obtain a list of workload occurrences in a database. A workload occurrence is a database connection with attributes that match a workload definition. You can list the workload occurrences for either a specific service class or for all service classes. For more information about the usage of this table function, see "Example: Investigating agent usage by service class" on page 171.
- WLM_GET_SERVICE_CLASS_AGENTS(*service_superclass_name*, *service_subclass_name*, *app_handle*, *dbpartitionnum*). Use this table function to obtain the list of agents working in the database. You can list all agents running in a specific service class or all agents working on behalf of a particular application. You can also use this table function to determine the state of the coordinator agent and subagents for applications and determine which requests each agent in the system is working on. For more information about the usage of this table function, see "Example: Investigating agent usage by service class" on page 171.
- WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(*app_handle*, *dbpartitionnum*). Use this table function to obtain a list of the active activities that are associated with a workload occurrence on a specific database partition (you can use wildcard characters to span application identifiers and database partitions). This table function returns information about activities that are queued, idle, or running. This table function does not return information about activities that have finished running. For more information about the usage of this table function, see "Example: Aggregating data using workload management table functions" on page 160 and "Example: Identifying hung activities" on page 167.
- WLM_GET_ACTIVITY_DETAILS(*app_handle*, *uow_id*, *activity_id*, *dbpartitionnum*). Use this table function to obtain detailed information about an activity in progress, which you identify by the unique combination of the activity identifier, unit of work identifier, and application identifier. You can use this table function to analyze information about activities that is returned by the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function. For more information about the usage of this table function, see "Example: Monitoring current system behavior at different levels using workload management table functions" on page 156.

Workload management table functions and snapshot monitor integration

You can use workload management table functions with the snapshot monitor table functions when performing problem determination or performance tuning.

The 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.

Table 15. Fields shared between the workload management and snapshot monitor table functions

Workload manager table function field	Snapshot monitor table function field
agent_tid	**agent_pid**

Table 15. Fields shared between the workload management and snapshot monitor table functions (continued)

Workload manager table function field	Snapshot monitor table function field
application_handle	agent_id agent_id_holding_lock
session_auth_id	session_auth_id
dbpartitionnum	node_number
utility_id	utility_id
workload_id	workload_id

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:

```
SELECT SUBSTR(UTILITY_TYPE,1,4) TYPE,
       UTILITY_PRIORITY PRIORITY,
       SUBSTR(UTILITY_DESCRIPTION,1,12) AS UTILITY_DESCRIPTION,
       SUBSTR(UTILITY_DBNAME,1,8) AS DBNAME,
       UTILITY_STATE,
       SUBSTR(UTILITY_INVOKER_TYPE,1,7) INVOKER,
       SUBSTR(CHAR(WLM.DBPARTITIONNUM),1,4) PART,
       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.DBPARTITIONNUM;
```

The output might resemble the following output:

```
TYPE PRIORITY    UTILITY_DESCRIPTION DBNAME   UTILITY_STATE INVOKER PART SUPERCLASS_NAME     SUBCLASS_NAME
---- ----------- ------------------- -------- ------------- ------- ---- ------------------- ------------------
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    1    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    1    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    1    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    2    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    2    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    2    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    3    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    3    BATCH               SYSDEFAULTSUBCLASS
LOAD           - OFFLINE LOAD        SAMPLE   EXECUTE       USER    3    BATCH               SYSDEFAULTSUBCLASS
```

Workload management stored procedures

You can use stored procedures for canceling an activity, capturing details about an activity, and resetting the statistics on workload management objects.

The following stored procedures are available:

- 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 canceled 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.

Note: If you are using this stored procedure to collect activity information for a procedure that has INOUT parameters, the INOUT values might be overwritten by the time that the capture occurs. This situation does not occur if you created the service class, workload, work action, or predictive threshold for which you are capturing activity data as COLLECT ACTIVITY WITH DETAILS AND VALUES, or if you alter the service class, workload, work action, or predictive threshold and specify the COLLECT ACTIVITY DATA keyword with either the ON COORDINATOR or ON ALL keyword, and with the WITH DETAILS AND VALUES keyword.

- WLM_COLLECT_STATS(). Use this stored procedure to collect and reset statistics for 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.

Workload management event monitors

Event monitors collect historical information or capture a set of events for debugging. By contrast, table functions collect and report point-in-time information.

You can use the following types of event monitors in a workload management configuration:

- ACTIVITIES. This type of event monitor captures information about individual activities. In some situations, you can have the event monitor include statement information and the data values used for input variables for SQL activities. You can use the activities collected by the activity event monitor as input into tools such as db2advis. You can also use an ACTIVITIES event monitor to capture information for debugging individual activities.

  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.
- THRESHOLD VIOLATIONS. This type of event monitor captures information each time that an activity violates a threshold. The information includes the identifier, unit of work, and application handle to uniquely identify the activity that violated the threshold, and the action that was applied to the activity (STOP EXECUTION or CONTINUE).

  If you specify COLLECT ACTIVITY DATA for the threshold and an activity 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. This type of event monitor captures statistics that are measured over a set period of time. Compared to statement or activity event monitors, the STATISTICS event monitor is an inexpensive method of capturing historical information because this type of event monitor deals with aggregated activity information instead of individual activities and you can target it to a single

service class or work class. See "Collecting workload management statistics using a statistics event monitor" on page 121 for a description of how to send statistics to the event monitor.

Unlike statement, connection, and transaction event monitors, the activity, statistics, and threshold violation 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.

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.

Statistics management

Statistics for workload management objects

Statistics are maintained for 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 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.

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 workload management statistics table functions report the current in-memory values, following a collection they report the reset values. The 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.

The following statistics are maintained on the given objects on each database partition, regardless of the value of the COLLECT AGGREGATE ACTIVITY DATA or the COLLECT AGGREGATE REQUEST DATA option specified for those objects when they are created or altered.

- Threshold queues:
  - Queue assignments total (**queue_assignments_total**). 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).
  - Queue size top (**queue_size_top**). Use this statistic to help determine the maximum queue size and to identify whether the queue size is sufficient.
  - Queue time total (**queue_time_total**). Use this statistic to determine how much time activities are spending in the queue and whether that time is excessive.
- Service subclasses:

  - Concurrent activity top (**concurrent_act_top**). Use this statistic to determine the highest concurrency of activities (including nested activities) reached on a database partition for a service class in the time interval for which the statistic is collected.
  - Coordinator activities completed total (**coord_act_completed_total**). Use this statistic to determine how much work is being performed in a service class.
  - Coordinator activities aborted total (**coord_act_aborted_total**). 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.
  - Coordinator activities rejected total (**coord_act_rejected_total**). Use this statistic, 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.
  - Number of active requests (**num_requests_active**). Use this statistic to determine the number of requests that are currently executing in a service class.
- Service superclasses:
  - Concurrent connection top (**concurrent_connection_top**). Use this statistic to tune a connection concurrency threshold.
- Workloads:
  - Workload occurrences completed total (**wlo_completed_total**). Use this statistic to determine how many occurrences of a workload complete in a specific period of time.
  - Concurrent workload occurrences top (**concurrent_wlo_top**). Use this statistic 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).
  - Concurrent activity top (**concurrent_act_top**). Use this statistic to tune the CONCURRENTWORKLOADACTIVITIES threshold.
  - Coordinator activities completed total (**coord_act_completed_total**). Use this statistic, which measures the rate of successful completion of activities, to obtain an indication of the health of the system.
  - Coordinator activities aborted total (**coord_act_aborted_total**). 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.
  - Coordinator activities rejected total (**coord_act_rejected_total**). 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.
  - Workload occurrences completed total (**wlo_completed_total**). Use this statistic to determine how many occurrences of a workload complete in a specific period of time.
- Work class (through a work action):
  - Activities total (**act_total**). 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.

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:

- Request execution time histogram. The execution time for requests is collected in a histogram for each database partition and for all requests. The request execution time approximates the effort spent by agents working on activities (which are composed of one or more requests). Use this information to understand where work is being performed and whether the distribution of work across partitions is uniform. (For example, coordinator activity counts might show that most activities originate on one database partition, but as part of processing the activities, the coordinator agent might be sending requests to another database partition that performs most of the work.) The request execution time histogram can be useful in determining the size of requests sent to a database partition (that is, whether the work that is sent to the database partition consists of mostly small requests or mostly large requests or whether there is no specific distribution).
- Request execution time average (**request_exec_time_avg**). Use this statistic to quickly understand the average amount of time that is spent processing each request on a database partition and to help tune the histogram template for the corresponding request execution time histogram.

When you set the value of the COLLECT AGGREGATE ACTIVITY DATA option to BASE for a service subclass or a work class (through a work action), the following statistics are collected or histograms are generated for each database partition for the corresponding service class or work class. 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.

- Average coordinator activity lifetime (**coord_act_lifetime_avg**). Use this statistic to determine the arithmetic mean of the lifetime for non-nested coordinator activities associated with a service class or a work class.
- Average coordinator activity execution time (**coord_act_exec_time_avg**). Use this statistic to determine the arithmetic mean of execution time for non-nested coordinator activities associated with a service class or a work class
- Average coordinator activity queue time (**coord_act_queue_time_avg**). Use this statistic to determine the arithmetic mean of the queue time for non-nested coordinator activities associated with a service class or a work class.
- Cost estimate top (**cost_estimate_top**). Use this statistic to tune estimated cost thresholds.
- Estimated rows returned top (**rows_returned_top**). Use the information to tune the actual rows returned thresholds.
- Temporary table space top (**temp_tablespace_top**). 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.
- Activity lifetime histogram. This histogram collects the time duration between the activity arrival and end time for non-nested coordinator activities. Use this histogram to obtain a view of overall system performance. 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.

- Activity execution time histogram. This histogram collects the execution time for non-nested coordinator activities. Use this histogram to measure the impact of changes to the system that affect execution time. 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.
- Activity queue time histogram. This histogram collects the amount of time that non-nested coordinator activities spend queued. Use this histogram to measure the impact of queueing thresholds on activities.

When you set the value of the COLLECT AGGREGATE ACTIVITY DATA option to EXTENDED for a service subclass or a work class, the following system statistics are collected or histograms are generated for each database partition 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.

Note: EXTENDED statistics are useful for more detailed performance modelling. See "Workload management performance modelling" on page 129.

- Non-nested coordinator activity inter-arrival time average (**coord_act_interarrival_time_avg**). 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 or work class and the next coordinator activity to arrive. The average is computed since the last statistics reset.
- Coordinator activity estimated cost average (**coord_act_est_cost_avg**). 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 or work class since the last statistics reset.
- Activity inter-arrival time histogram. This histogram collects the inter-arrival time for non-nested coordinator activities. Use this histogram to obtain the inter-arrival time distribution for non-nested coordinator activities. This data is useful for modelling your system or for inputting into performance-modeling applications.
- Activity estimated cost histogram. This histogram collects the estimated cost for non-nested coordinator activities. Use this histogram to obtain an approximate service time distribution. This data is useful for modelling your system or for inputting into performance-modelling applications.

The following table provides a quick reference of which statistics are collected for each workload management object. Some statistics are always collected for each object. 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 16. Statistics collected for each workload management object

Object type	Statistic collected by default	Statistics collected when you specify COLLECT AGGREGATE ACTIVITY DATA BASE	Statistics collected when you specify COLLECT AGGREGATE ACTIVITY DATA EXTENDED	Statistics collected when you specify COLLECT AGGREGATE REQUEST DATA BASE
Threshold queue	• **queue_assignments_total** • **queue_size_top** • **queue_time_total**	N/A	N/A	N/A
Service subclass	• **coord_act_completed_total** • **coord_act_rejected_total** • **coord_act_aborted_total** • **concurrent_act_top** • **num_requests_active**	• **cost_estimate_top** • **rows_returned_top** • **temp_tablespace_top** • **coord_act_lifetime_top** • **request_exec_time_avg** • **coord_act_lifetime_avg** • **coord_act_exec_time_avg** • **coord_act_queue_time_avg** • Activity lifetime histogram • Activity execution time histogram • Activity queue time histogram • Request execution time histogram	• **coord_act_est_cost_avg** • **coord_act_interarrival_time_avg** • Activity inter-arrival time histogram • Activity estimated cost histogram	• **request_exec_time_avg** • Request execution time histogram
Service superclass	• **concurrent_connection_top**	N/A	N/A	
Workload	• **concurrent_wlo_top** • **concurrent_act_top** • **coord_act_completed_total** • **coord_act_rejected_total** • **coord_act_aborted_total** • **wlo_completed_total**	N/A	N/A	
Work class (through a work action)	• **act_total**	• **cost_estimate_top** • **rows_returned_top** • **temp_tablespace_top** • **coord_act_lifetime_top** • **coord_act_lifetime_avg** • **coord_act_exec_time_avg** • **coord_act_queue_time_avg** • Activity lifetime histogram • Activity execution time histogram • Activity queue time histogram	• **coord_act_est_cost_avg** • **coord_act_interarrival_time_avg** • Activity inter-arrival time histogram • Activity estimated cost histogram	

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 41 bins. The number of bins is fixed. The 40th bin contains the highest defined value for the histogram, while the 41st bin is for values that are beyond the highest defined value. The following figure shows a histogram of activity lifetimes plotted to a bar chart:

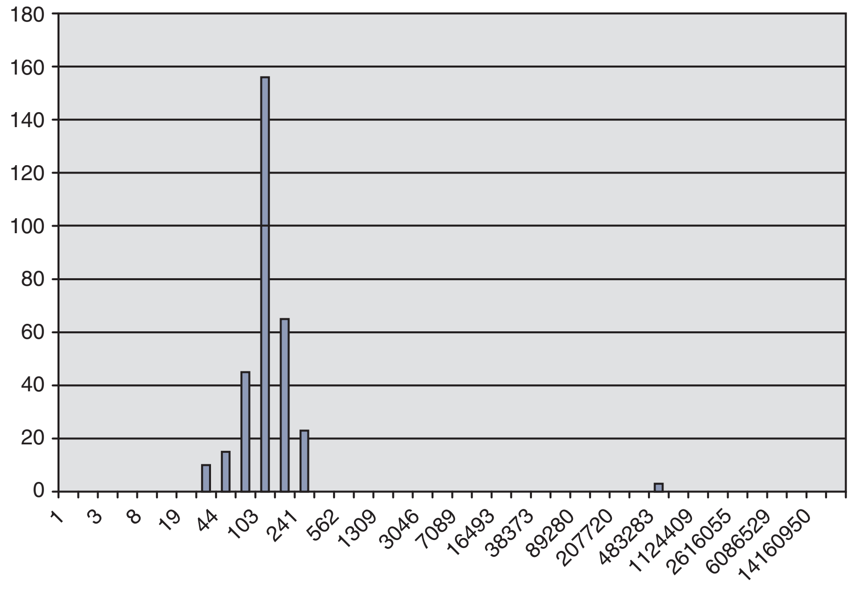


Figure 24. Histogram plotted to a bar chart

The plotted activity lifetime histogram corresponds to the following data. Each count represents the number of activities whose lifetime (in milliseconds) fell within the range of the low bin value to the high bin values. For example, 156 activities had a lifetime in the range 68 milliseconds to 103 milliseconds.

```
Low Bin  High Bin Count
      0         1     0
      1         2     0
      2         3     0
      3         5     0
      5         8     0
      8        12     0
     12        19     0
     19        29    10
     29        44    15
     44        68    45
     68       103   156
    103       158    65
    158       241    23
    241       369     0
    369       562     0
    562       858     0
    858      1309     0
   1309      1997     0
   1997      3046     0
   3046      4647     0
   4647      7089     0
   7089     10813     0
  10813     16493     0
  16493     25157     0
  25157     38373     0
  38373     58532     0
  58532     89280     0
  89280    136181     0
 136181    207720     0
 207720    316840     0
 316840    483283     3
 483283    737162     0
 737162   1124409     0
1124409   1715085     0
1715085   2616055     0
```

```
 2616055   3990325     0
 3990325   6086529     0
 6086529   9283913     0
 9283913  14160950     0
14160950  21600000     0
21600000  Infinity     0
```

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 “Example: Tuning a workload management configuration when capacity planning information is unavailable” on page 173 and “Example: Computing averages and a standard deviation from histograms in a workload management configuration” on page 161 for examples of how to use histograms to better understand and characterize your workload.

In a partitioned database environment, histograms are collected on each database partition. Histogram bins have the same values on all database partitions. You can use the bins to analyze information on a per partition basis. You can also combine the histograms from all database partitions by adding the counts in the corresponding bins into a 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 from the global histogram.

Histograms are available for service subclasses and work classes (through work actions). Histograms are collected for these objects when you specify one of the COLLECT AGGREGATE ACTIVITY DATA clauses when creating or altering the object. For work classes, histograms are collected if a COLLECT AGGREGATE ACTIVITY DATA work action is applied 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 the service subclass or for a work action applied to the work class).
- Non-nested coordinator activity execution time (when you specify AGGREGATE ACTIVITY DATA BASE or AGGREGATE ACTIVITY DATA EXTENDED for the service subclass or for a work action applied to the work class).
- Non-nested coordinator activity queue time (when you specify AGGREGATE ACTIVITY DATA BASE or AGGREGATE ACTIVITY DATA EXTENDED for the service subclass or for a work action applied to the work class).
- Request execution time (when you specify AGGREGATE REQUEST DATA BASE for the service class). This histogram does not apply to work classes.
- Non-nested activity inter-arrival time histogram (when you specify AGGREGATE ACTIVITY DATA EXTENDED for the service subclass or for a work action applied to the work class).
- Non-nested DML activity estimated cost (when you specify AGGREGATE ACTIVITY DATA EXTENDED for the service subclass or for a work action applied to the work class).

All activity-related histograms collect activities that complete, abort, or are rejected.

You can optionally specify a histogram template that describes the high bin values for each of the histograms that are collected for an object. Histogram templates are used to determine the high bin values for a histogram. Histogram templates are *unitless* objects that specify what a particular histogram should look like (unitless meaning that there is no predefined measurement unit assigned to the histogram template).

You apply a histogram template by using the appropriate HISTOGRAM TEMPLATE keyword when creating or altering service subclasses or work actions. If you do not specify a histogram template, the default template SYSDEFAULTHISTOGRAM is used. If AGGREGATE ACTIVITY DATA collection is not enabled for an object, the histogram template is ignored.

You can create a histogram template by using the CREATE HISTOGRAM TEMPLATE statement and specifying the maximum high bin value. All other bins 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, you would 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:

```
Low Bin High Bin
      0        1
      1        2
      2        3
      3        4
      4        6
      6        9
      9       13
     13       19
     19       28
     28       41
     41       60
     60       87
     87      127
    127      184
    184      268
    268      389
    389      565
    565      821
    821     1192
   1192     1732
   1732     2514
   2514     3651
   3651     5300
   5300     7696
   7696    11173
  11173    16222
  16222    23553
  23553    34196
  34196    49649
  49649    72084
  72084   104657
 104657   151948
 151948   220609
 220609   320297
 320297   465030
 465030   675163
 675163   980250
 980250  1423197
1423197  2066299
2066299  3000000
3000000 Infinity
```

You can then use this histogram template for an existing histogram. For example, to use the TEMPLATE1 histogram template for the activity lifetime histogram of service subclass MYSUBCLASS under the service superclass MYSUPERCLASS, issue a statement such as the following one:

```
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 this service subclass has high bin values that are determined by the TEMPLATE1 histogram template instead of using the default high bin values from the SYSDEFAULTHISTOGRAM histogram template.

Note: If you change a service class 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 the corresponding histogram template high bin values by querying the SYSCAT.HISTOGRAMTEMPLATEBINS view (the low bin value is always the high bin value from the preceding bin, or 0 for the first bin).

Collecting workload management statistics using a statistics event monitor

Statistics for workload management objects can be sent to a statistics event monitor for historical analysis.

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 workload management objects” on page 113 for a description of which statistics are collected for each 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.

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:
   ```
   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. However, only one event monitor of the STATISTICS type can be active on a database partition at one time. 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 workload management object. See "Statistics for workload management objects" on page 113 for details on which statistics are collected by default for each object. Specify the collection of aggregate activity data for service subclasses 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 **wlm_collect_int**. The **wlm_collect_int** parameter specifies an interval of time in minutes. Every interval, the in-memory copy of the workload management statistics for all workload management objects is written to the active statistics event monitor and the in-memory statistics are reset. In a partitioned database environment, the **wlm_collect_int** parameter must be updated on the catalog partition. This parameter can be updated dynamically. For example:
   ```
   CONNECT TO database alias
   UPDATE DATABASE CONFIGURATION USING WLM_COLLECT_INT 5 IMMEDIATE
   ```

After you perform the preceding steps, workload management statistics are written to the statistics event monitor every **wlm_collect_int** minutes. Each record written to the statistics event monitor has a STATISTICS_TIMESTAMP value and a LAST_WLM_RESET value. The interval of time from LAST_WLM_RESET to STATISTICS_TIMESTAMP defines the collection interval (that is, interval of time over which the statistics in that record were collected).

If the **wlm_collect_int** parameter is set to a nonzero value and there is no active statistics event monitor, the in-memory workload management statistics are still reset every **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 **wlm_collect_int** value without activating a statistics event monitor.

If the **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 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 in-memory statistics are sent to the statistics event monitor and the in-memory statistics are reset. If there is no active statistics event monitor, the in-memory values are reset, but data is not collected. If you only want to reset statistics, you can invoke the 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 **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 5:30 AM. You can invoke the WLM_COLLECT_STATS procedure at 5:55 AM, which sends the in-memory values of the statistics to the event monitor and resets the statistics. The next automatic statistics collection still occurs at 6:30 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.

Note: The workload management statistics table functions report the current values of the in-memory 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 in-memory 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.

Note: 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.

Note: When a database is deactivated, the in-memory 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.

Note: 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 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 manager statistics applies to all users.

Workload management table functions to obtain statistics

You can use the table functions described in this topic to obtain statistics about workload management objects.

The workload management table functions are available in the SYSPROC schema. They are high performance and can return information about the work occurring in the system with very little impact on currently executing workloads.

You can use statistics for a number of different purposes, such as verifying whether changes to the workload management configuration have had the expected effect. For example, if you create a new work class to classify READ activities, you might want to verify that READ activities are being classified under the new work class. 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.

All statistics table functions return the statistics that accumulated since the last time that the statistics were reset.

For the *dbpartitionnum* variable, you can specify -2 to indicate that you want to collect data from all database partitions or -1 to indicate that you want to collect data only from the database partition to which the application that issued the table function call is connected (that is, the coordinator partition). If you call these table functions from application programs, you should use the SQLM_CURRENT_NODE and SQLM_ALL_NODE constants in the sqlmon.h header file to avoid using -1 and -2 as literals.

- WLM_GET_SERVICE_SUPERCLASS_STATS(*service_superclass_name*, *dbpartitionnum*). Use this table function to obtain information about the concurrent connection high watermark that was calculated since the last statistics reset. You can use wildcard characters to span service superclasses and database partitions.
- WLM_GET_SERVICE_SUBCLASS_STATS(*service_superclass_name*, *service_subclass_name*, *dbpartitionnum*). Use this table function to obtain summarized statistics such as the number of activities and average execution time calculated since the last statistics reset. You can obtain this information for multiple service subclasses across one or more database partitions. You can use wildcard characters to span service superclasses, service subclasses, and database partitions. For more information about the usage of this table function, see "Example: Obtaining point-in-time statistics from service classes" on page 159, "Example: Aggregating data using workload management table functions" on page 160, "Example: Analyzing a service class–related system slowdown" on page 163, and "Example: Investigating a workload-related system slowdown" on page 165.
- WLM_GET_WORKLOAD_STATS(*workload_name*, *dbpartitionnum*). Use this table function to obtain summarized statistics for one or all workloads and database partitions. You can use wildcard characters to span workload names and database partitions. The statistics returned include information about completed activities and high watermark information that were computed since the statistics reset. For more information about the usage of this table function, see "Example: Investigating a workload-related system slowdown" on page 165.
- WLM_GET_WORK_ACTION_SET_STATS(*work_action_set_name*, *dbpartitionnum*). Use this table function to obtain summarized statistics for one or more work action sets across one or more database partitions. The statistics returned include information about how many activities were assigned to each work class since the last statistics reset. For information about the usage of this table function, see "Example: Analyzing workloads by activity type" on page 166.
- WLM_GET_QUEUE_STATS (*threshold_predicate*, *threshold_domain*, *threshold_name*, *threshold_id*). Use this table function to obtain information about threshold queues. This information tells you how many activities are currently queued and the last time that an activity left a queue. Statistical information is also included, such as the average time that an activity spent in the queue and the high watermark for the queue size. You can use this information when determining whether a specific queue is becoming a bottleneck in the system (that is, activities are spending too much time in this queue).

Resetting statistics on workload management objects

This topic describes how to reset statistics for workload management objects.

Four events will reset the in-memory statistics stored for each workload management object. (For a description of the statistics maintained for each object, see "Statistics for workload management objects" on page 113.)

- The WLM_COLLECT_STATS stored procedure is invoked. See “Collecting workload management statistics using a statistics event monitor” on page 121 for details.
- The automatic 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 121 for details.
- The database is reactivated. Every time the database is activated on a database partition, the statistics for all workload management objects on that database partition 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 is committed, the in-memory statistics for that service subclass are reset.

You can determine the last time the statistics were reset for a given 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 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 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 threshold violations

When a workload management threshold is violated, a threshold violation record is written to the active THRESHOLD VIOLATIONS event monitor, if one exists.

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 allowed to continue or was stopped. If the activity was stopped, the application that submitted the activity will have received an SQL4712N error.

You can optionally have detailed activity information (including statement text) written to an active activity 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.

To monitor threshold violations:

1. Use the CREATE EVENT MONITOR statement to create an event monitor of type THRESHOLD VIOLATIONS. For example:
   ```
   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. However, only one event monitor of the THRESHOLD VIOLATIONS type can be active on a database partition at one time. 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.

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).

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 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

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.

To enable collection of activities for a given 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. However, only one event monitor of the ACTIVITIES type can be active on a database partition at one time. 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.

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 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.

If you are using the WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure to collect activity information for a procedure that has INOUT parameters, the INOUT values might be overwritten by the time that the capture occurs. This situation does not occur if the activity is collected as a result of the COLLECT ACTIVITY DATA WITH DETAILS AND VALUES keyword on a service subclass, workload, work action, or predictive threshold.

Importing activity information into the Design Advisor

You can import activities collected by an activity event monitor into the Design Advisor to help you make decisions about the database objects accessed by these activities.

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 activity 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 activity event monitor tables through the Design Advisor command line interface.

Cancelling activities

If an activity is consuming too many resources, or is hung, you may want to cancel it. Cancelling an activity is a gentler approach than forcing the application that submitted the activity. A cancelled activity returns SQL4725N to the user, but does not end the connection or affect any other activities of that user. Forcing the application ends both the connection and activities of that user.

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 cancelled 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 cancellable, including the load utility and stored procedures.

To cancel an activity:

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 WLM_GET_ACTIVITY_DETAILS 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 "Example: Identifying hung activities" on page 167.

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 like the following threshold:

```
CREATE THRESHOLD LONGRUNNINGACTIVITIES FOR DATABASE ACTIVITIES ENFORCEMENT DATABASE
WHEN ACTIVITYTOTALTIME > 3 HOURS COLLECT ACTIVITY DATA WITH DETAILS CONTINUE
```

You can then analyze the information that is written to the event monitor. DML activities also have their statement text and compilation environment information written to the 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 CPU 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. See "Statistics for workload management objects" on page 113 for more information.

Working with histograms

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.

To create a histogram template, you require SYSADM or DBADM authority.

See the following topics for more information about prerequisites:

- Appendix A, "Workload management DDL statement considerations," on page 267
- Naming rules

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 117.

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.

You require SYSADM or DBADM authority to alter a histogram template.

See Appendix A, "Workload management DDL statement considerations," on page 267 for more information about prerequisites.

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 117.

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 workload management objects” on page 124 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.

To drop a histogram template, you require SYSADM or DBADM authority.

See Appendix A, “Workload management DDL statement considerations,” on page 267 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 or a work action. You can view the service subclasses and work actions that reference a histogram template by querying the SYSCAT.HISTOGRAMTEMPLATESUSE view.

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.

Part 4. Examples

Chapter 7. Workload management examples

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 the 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 migrating to DB2 Version 9.5, 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. WL_ENGINEERING 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:

Table 17. Service class definitions

Service class
SALES
ACCOUNTING
ENGINEERING
TESTING
PRODUCTION
SYSDEFAULTUSERCLASS
SYSDEFAULTMAINTENANCECLASS
SYSDEFAULTSYSTEMCLASS

With the workload management solution implemented as described above, 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 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 allow for 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 TOTALSCPARTITIONCONNECTIONS > 300 STOP EXECUTION
```

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

Table 18. Configuration after adding threshold for the TESTING service superclass

Service class	MAXSERVICECLASSCONNECTIONS threshold
SALES	N/A
ACCOUNTING	N/A
ENGINEERING	N/A
TESTING	300
PRODUCTION	N/A
SYSDEFAULTUSERCLASS	N/A

Table 18. Configuration after adding threshold for the TESTING service superclass (continued)

Service class	MAXSERVICECLASSCONNECTIONS threshold
SYSDEFAULTMAINTENANCECLASS	N/A

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 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 workload management configuration is as follows:

Table 19. Configuration after changing prefetch priority for the TESTING service superclass

Service class	MAXSERVICECLASSCONNECTIONS threshold	Prefetch priority
SALES	N/A	DEFAULT
ACCOUNTING	N/A	DEFAULT
ENGINEERING	N/A	DEFAULT
TESTING	300	LOW
PRODUCTION	N/A	DEFAULT
SYSDEFAULTUSERCLASS	N/A	DEFAULT
SYSDEFAULTMAINTENANCECLASS	N/A	DEFAULT

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 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 TOTALDBPARTITIONCONNECTIONS > 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 workload management configuration is as follows:

Table 20. Configuration with EXPERIMENT service subclass

Service class	**MAXSERVICECLASSCONNECTIONS threshold**	**Prefetch priority**
SALES	N/A	DEFAULT
ACCOUNTING	N/A	DEFAULT
ENGINEERING	N/A	DEFAULT
EXPERIMENT	50	LOW
TESTING	300	LOW
PRODUCTION	N/A	DEFAULT
SYSDEFAULTUSERCLASS	N/A	DEFAULT
SYSDEFAULTMAINTENANCECLASS	N/A	DEFAULT

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: 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.

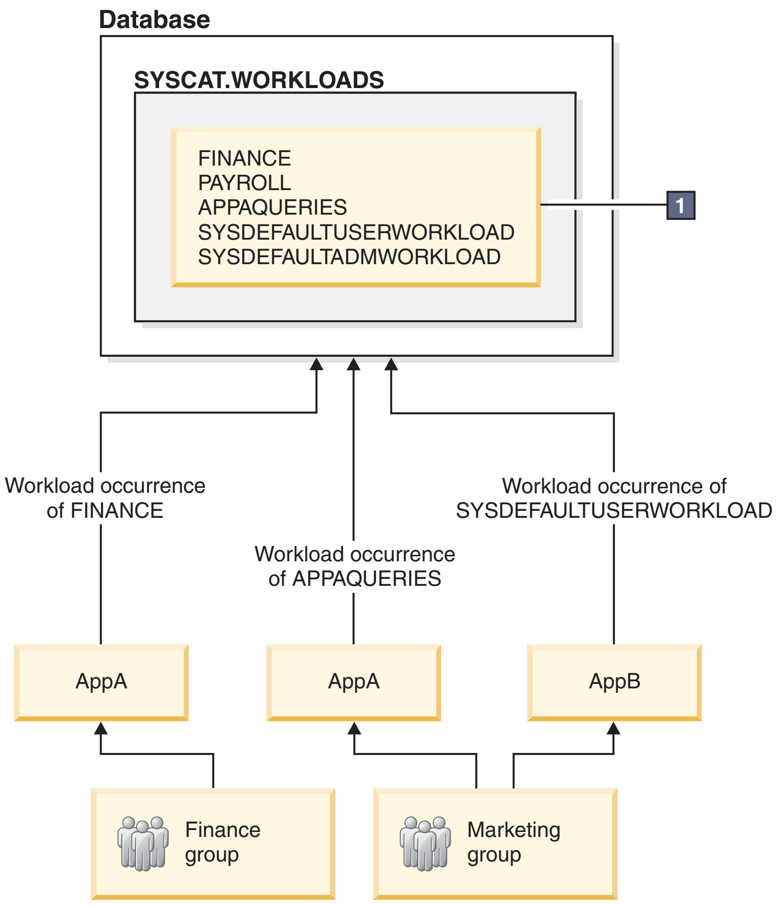


Figure 25. Example of workload assignment

1 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
```

In a three-tier client/server environment, the database connection is established by the application server that is working on behalf of the clients. The application server can use the sqleseti (set client information) API to pass client information to the DB2 data server; otherwise, only the information from the application server is passed, and that information is likely to be the same for all client requests that are routed through this application server. When the data server assigns units of work from different clients to different workloads (and to different service classes), the data server uses the client information attributes (that is, the client user ID, client application name, client workstation name, and client accounting string) as criteria for associating a unit of work with a workload.

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`.

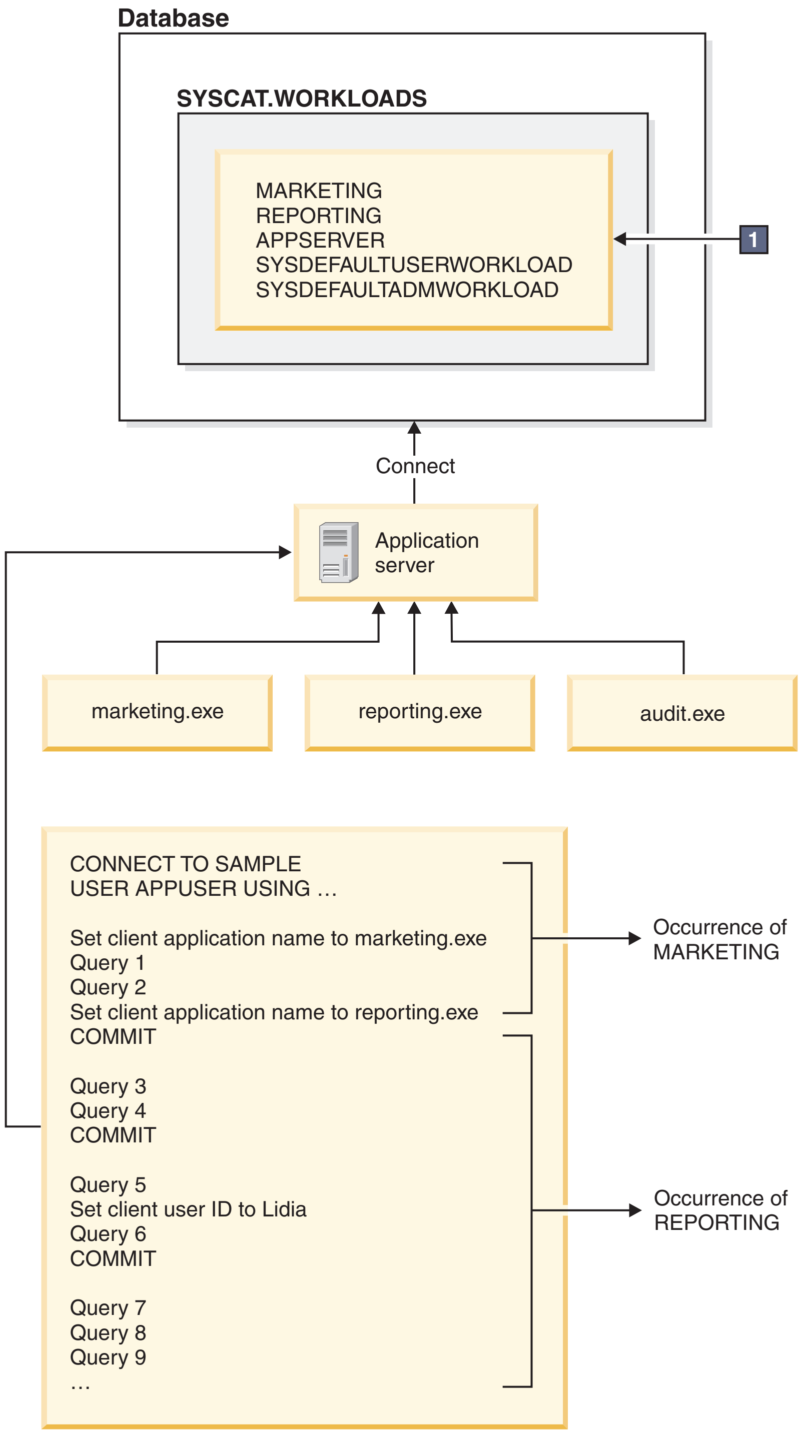


Figure 26. Example of workload assignment in a three-tier environment

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 SC1
```

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 21. Workloads in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
1	REPORTS	AppA								
2	INVENTORY REPORT	AppB	LYNN		ACCOUNTING	TELEMKTR				
3	SALES REPORT	AppC	KATE	KATE		SALESREP				
4	AUDIT REPORT	AppB			ACCOUNTING	FINANALYST				
5	EXPENSE REPORT	AppA	TIM			EXPENSE APPROVER				
6	AUDIT RESULT			LYNN			LYNN			Audit Group

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

Table 22. Database connection attributes

APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
AppA	TIM	TIM	FINANCE	FINANALYST, EXPENSE APPROVER	NULL	NULL	NULL	Business account

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 23. REPORTS workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
1	REPORTS	AppA								

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_WRKSTNNAME, 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 24. Workloads in the catalog after repositioning the EXPENSEREPORT workload

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
1	EXPENSE REPORT	AppA	TIM			EXPENSE APPROVER				
2	REPORTS	AppA								
3	INVENTORY REPORT	AppB	LYNN		ACCOUNTING	TELEMKTR				
4	SALES REPORT	AppC	KATE	KATE		SALESREP				
5	AUDIT REPORT	AppB			ACCOUNTING	FINANALYST				
6	AUDIT RESULT			LYNN			LYNN			Audit Group

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 25. Workloads in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
1	EXPENSE REPORT	AppB	TIM			EXPENSE APPROVER				
2	REPORTS	AppB								
3	INVENTORYREPORT	AppA	LYNN		ACCOUNTING	TELEMKTR				
4	SALES REPORT	AppC	KATE	KATE		SALESREP				
5	AUDIT REPORT	AppA			ACCOUNTING	FINANALYST				
6	AUDIT RESULT			LYNN			LYNN			Audit Group

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

Table 26. Database connection attributes

APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
AppA	LYNN	LYNN	ACCOUNTING	FINANALYST, SALESREP	LYNN	NULL	wrkstn2	Audit group

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 EXPENSEREPORT workload:

Table 27. EXPENSEREPORT workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
1	EXPENSEREPORT	AppB	TIM			EXPENSE APPROVER				

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 28. REPORTS workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
2	REPORTS	AppB								

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 29. INVENTORYREPORT workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
3	INVENTORYREPORT	AppA	LYNN		ACCOUNTING	TELEMKTR				

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 30. SALESREPORT workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
4	SALESREPORT	AppC	KATE	KATE		SALESREP				

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 31. AUDITREPORT workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
5	AUDITREPORT	AppA			ACCOUNTING	FINANALYST				

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 32. AUDITRESULT workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURRENT CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
6	AUDITRESULT			LYNN			LYNN			Audit Group

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 allow more than one value for a connection attribute.

Assume that the following workloads are defined in the catalog:

Table 33. Workloads in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURREN CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
1	ITEMINQ		KYLE, GEORGE		RETAIL, SALES					
2	DAILY TRANS REPORT	AppC		KYLE, CAROL	SALES, ACCOUNTING					
3	SALES SUMMARY	AppA, AppB				ACCOUNTANT, FINANALYST				

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

Table 34. Database connection attributes

APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURREN CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
AppC	LINDA	KYLE	SALES	ACCOUNTANT	LINDA	NULL	NULL	Business Account

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 35. ITEMINQ workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURREN CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
1	ITEMINQ		KYLE, GEORGE		RETAIL, SALES					

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 ITEMNO workload values are `KYLE` and `GEORGE`. No match occurs for this attribute.

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

Table 36. DAILYTRANSREPORT workload in the catalog

Evaluation order	Workload name	APPLNAME	SYSTEM _USER	SESSION _USER	SESSION _USER GROUP	SESSION _USER ROLE	CURRENT CLIENT _USERID	CURRENT CLIENT _APPLNAME	CURREN CLIENT _WRKSTNNAME	CURRENT CLIENT _ACCTNG
2	DAILYTRANSREPORT	AppC		KYLE, CAROL	SALES, ACCOUNTING					

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.

Example: Using thresholds

You can use thresholds for a variety of purposes. In the scenario in this topic, thresholds are used to control the number of large jobs running, allow for different execution times for different applications, and control the behavior of an application that is in development.

One way of setting up a workload management solution is to divide and manage the database resources across 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.

In this situation, you create a workload definition for each one of these applications to map the application to its applicable service superclass. The database catalogs 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.

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.

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 running in an hour or less, you create a threshold with a database domain, disallowing 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 create a new service subclass under the Sales service superclass and call it YearlySalesReports. You then map the YearlySales workload to this service subclass, create an activity total time threshold of 5 hours, and associate this threshold with the service subclass. The new value of 5 hours now applies to the YearlySalesReports service subclass, although the global value of 1 hour applies elsewhere in the database.

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.

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.

Example: CONCURRENTWORKLOADOCCURRENCES, TOTALDBPARTITIONCONNECTIONS, and TOTALSCPARTITIONCONNECTIONS thresholds

The example in this topic shows how the CONCURRENTWORKLOADOCCURRENCES, TOTALDBPARTITIONCONNECTIONS, and TOTALSCPARTITIONCONNECTIONS aggregate thresholds interact.

Assume that a database connection is established on database partition 1. The TOTALDBPARTITIONCONNECTIONS threshold is evaluated once to decide whether this connection is allowed to connect to the database. When the evaluation of this threshold is successful, TOTALDBPARTITIONCONNECTIONS is never evaluated again for the same connection, which is now in the database. The connection switches between workload A and B several times because the relevant connection attributes were changed. Each time that the connection switches workloads, it also potentially moves to a different service class. When an occurrence of either workload is started, the CONCURRENTWORKLOADOCCURRENCES threshold is evaluated. If this threshold is not violated, the connection then joins the corresponding service class, at which point the last threshold, TOTALSCPARTITIONCONNECTIONS, is evaluated. At this point the connection might be queued before entering the service class, before it can issue any work.

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 CALL and 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. You also want to be informed when the number of CALL statements that call any routine in the MYSCHEMA schema is greater than 5.

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

1. Create a MYWORKCLASSSET work class set that contains three work classes: one for queries with a large estimated cost, one for queries with a large estimated cardinality, and one for CALL statements that call procedures in the MYSCHEMA schema. 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,
WORK CLASS CALLSTATEMENTS WORK TYPE CALL ROUTINES IN SCHEMA MYSCHEMA)
```

2. Create a DATABASEACTIONS work action set that contains three 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,
WORK ACTION FIVECALLS ON WORK CLASS CALLSTATEMENTS
WHEN CONCURRENTDBCOORDACTIVITIES > 5 COLLECT ACTIVITY DATA CONTINUE)
```

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 and XQuery 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.

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. DDLACTIVITY, which is for all DDL activity
4. 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 DDLACTIVITY WORK TYPE DDL,
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 37. SERVICECLASSACTIONS work action set

Work action	**Work class applied to**	**Action**
MAPDML	SMALLDML	Maps to the SMALLACTIVITY service subclass
MAPOTHER	ALLACTIVITY	Maps to the OTHERACTIVITY service subclass
COUNTLOAD	LOADUTIL	Counts the number of LOAD activities

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 MAPOTHER ON WORK CLASS ALLACTIVITY MAP ACTIVITY TO OTHERACTIVITY,
WORK ACTION COUNTLOAD ON WORK CLASS LOADUTIL COUNT ACTIVITY)
```

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. If no work action were applied to the LOADUTIL work class, LOAD activities would fall under the ALLACTIVITY work class and have the MAPOTHER work action applied to them (when a work class does not have a work action applied to it, no activity is classified under that work class).

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 allowed 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 ESTIMATEDSQLCOST > 1000000 COLLECT ACTIVITY DATA STOP EXECUTION)
```

Table 38. DATABASEACTIONS work action set

Work action	Work class applied to	Threshold type and value	Action
CONCURRENTSMALLDML	SMALLDML	Concurrency up to 1000 statements; queue up to 10 000 statements	• Stop execution • Collect activity data
CONCURRENTLOAD	LOADUTIL	Concurrency up to 2 occurrences; queue up to 10 occurrences	• Stop execution • Collect activity data
CONCURRENTOTHER	ALLACTIVITY	Concurrency up to 100 activities; queue up to 100 activities	• Stop execution • Collect activity data
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 allowed 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 allowed 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 allowed 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.

Example: Using a 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 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
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

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 allow 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 with the ACTIVITYTOTALTIME threshold specified for each work class as follows:

Table 39. ACTIVITYTOTALTIME threshold specified for each work class

Work action	Work class applied to	Threshold type and value	Actions
SMALLDMLTIMEALLOWED	SMALLDML	ACTIVITYTOTALTIME < 31 MINUTES	• Stop execution • Collect activity data
MEDDMLTIMEALLOWED	MEDDML	ACTIVITYTOTALTIME < 31 MINUTES	• Stop execution • Collect activity data
LARGEDMLTIMEALLOWED	LARGEDML	ACTIVITYTOTALTIME < 5 HOURS	• Stop execution • Collect activity data
ALLDDLTIMEALLOWED	ALLDDL	ACTIVITYTOTALTIME < 31 minutes	• Stop execution • Collect activity data
ALLACTIVITYTIMEALLOWED	ALLACTIVITY	ACTIVITYTOTALTIME < 31 minutes	• Stop execution • Collect activity data

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 40. LARGEDMLTIMEALLOWED work action applied to the LARGEDML work class

Work action	Work class applied to	Threshold type and value	Action
LARGEDMLTIMEALLOWED	LARGEDML	ACTIVITYTOTALTIME < 5 HOURS	• Stop execution • Collect activity data

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 do not get classified and will have the ACTIVITYTOTALTIME database time threshold of less than 31 minutes applied to them.

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 workload management monitoring features, you can determine the different types of work running on your system, and the distribution of the work.

To accomplish this task, 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 WRITEWC 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 CLASSREADWC COUNT ACTIVITY,
WORK ACTION COUNTWRITE ON WORK CLASS WRITEWC 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(DBPARTITIONNUM),1,4) AS PART,
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, PART
```

Example: Monitoring current system behavior at different levels using workload management table functions

The workload management facility provides a variety of table functions that you can use to obtain data about your workload management configuration.

Installing DB2 Version 9.5 creates a set of default workloads and service classes. Before deciding how to implement your own 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 *dbpartitionnum*:

```
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,
       SUBSTR(CHAR(COORD_PARTITION_NUM),1,4) AS COORDPART,
       SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHNDL,
       SUBSTR(CHAR(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('', '', -2)) AS SCINFO
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART, APPHNDL, WORKLOAD_NAME, WLO_ID
```

Assume that the system has four database partitions and that there are two applications performing activities on the database when you issue the query. The results would resemble the following ones:

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART COORDPART APPHNDL WORKLOAD_NAME          WLO_ID
------------------- ------------------ ---- --------- ------- -----------------------------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         1       SYSDEFAULTUSERWORKLOAD 1
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    0         2       SYSDEFAULTUSERWORKLOAD 2
```

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 partition (partition 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:

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHNDL,
       SUBSTR(CHAR(WORKLOAD_NAME),1,22) AS WORKLOAD_NAME,
       SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,6) AS WLO_ID,
       SUBSTR(CHAR(SYSTEM_AUTH_ID),1,9) AS SYSAUTHID,
       SUBSTR(CHAR(APPLICATION_NAME),1,15) AS APPLNAME
FROM TABLE(WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES('', '', 0)) AS SCINFO
ORDER BY APPHNDL, WORKLOAD_NAME, WLO_ID

APPHNDL WORKLOAD_NAME          WLO_ID SYSAUTHID APPLNAME
------- ---------------------- ------ --------- ---------------
1       SYSDEFAULTUSERWORKLOAD 1      LYNN      accountspay
2       SYSDEFAULTUSERWORKLOAD 2      KATE      businessobjects
```

Then, you can use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table function to show the current activities of one of the workload occurrences:

```
SELECT SUBSTR(CHAR(COORD_PARTITION_NUM),1,5) AS COORD,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       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,1,9) AS ACTTYPE,
       SUBSTR(CHAR(NESTING_LEVEL),1,7) AS NESTING
FROM TABLE(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2)) AS WLOACTS
ORDER BY PART, UOWID, ACTID

COORD PART UOWID ACTID PARUOWID PARACTID ACTTYPE  NESTING
----- ---- ----- ----- -------- -------- -------- -------
0     0    1     3     -        -        CALL     0
0     0    1     5     1        3        READ_DML 1
```

```
0     1    1     5     -        -        READ_DML 1
0     2    1     5     -        -        READ_DML 1
0     3    1     5     -        -        READ_DML 1
```

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 partitions 0, 1, 2, and 3. The parent identifier information is available only on the coordinator partition.

You can obtain more information about an individual activity that is currently running by using the WLM_GET_ACTIVITY_DETAILS table function. This table function returns activity information as name-value pairs for each database partition. In the following example, the table function only shows an 11-member subset of the name-value pairs for each database partition. For a complete list of name-value pairs, see WLM_GET_ACTIVITY_DETAILS table function.

```
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

PART NAME                 VALUE
---- -------------------- ------------------------------
0    APPLICATION_HANDLE   1
0    COORD_PARTITION_NUM  0
0    UOW_ID               1
0    ACTIVITY_ID          5
0    PARENT_UOW_ID        1
0    PARENT_ACTIVITY_ID   3
0    ACTIVITY_TYPE        READ_DML
0    NESTING_LEVEL        0
0    INVOCATION_ID        1
0    ROUTINE_ID           0
0    LOCAL_START_TIME     2005-11-25-18.52.49.343000
1    APPLICATION_HANDLE   1
1    COORD_PARTITION_NUM  0
1    UOW_ID               1
1    ACTIVITY_ID          5
1    PARENT_UOW_ID        -
1    PARENT_ACTIVITY_ID   -
1    ACTIVITY_TYPE        READ_DML
1    NESTING_LEVEL        0
1    INVOCATION_ID        1
1    ROUTINE_ID           0
1    LOCAL_START_TIME     2005-11-25-18.52.49.598000
```

The three 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 *dbpartitionnum*:

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       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, PART, AGENT_TID

APPHANDLE PART AGENT_TID AGENTTYPE   AGENTSTATE REQTYPE      UOW_ID ACT_ID
--------- ---- --------- ----------- ---------- --------------------------
1         0    3         COORDINATOR ACTIVE     FETCH             1      5
1         0    4         PDBSUBAGENT ACTIVE     SUBSECTION:1      1      5
1         1    2         PDBSUBAGENT ACTIVE     SUBSECTION:2      1      5
```

The results show a coordinator agent and a subagent on database partition 0 and a subagent on database partition 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 partitions 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, *dbpartitionnum*, is -2 (a wildcard character), which means that data from all database partitions is to be returned.

Note: Lifetime information is only returned for those service classes that are defined with COLLECT AGGREGATE ACTIVITY DATA.

```
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 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, PART
```

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART 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
```

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 database partition:

```
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,
       CONCURRENT_ACT_TOP AS ACTHIGHWATERMARK
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '',  -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART

SUPERCLASS_NAME     SUBCLASS_NAME      PART 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 database partition 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 workload management table functions

You can perform various aggregations on table data in a workload management configuration to monitor the system and identify potential problems.

The following are examples of data aggregation that you can perform to identify problems.

Identifying increases in average query lifetimes because queries are spending too much time in the queue

You can identify a situation in which the average query lifetime increases because queries are spending too much time in the queue by showing the average time in the queue for coordinator activities for each service class, across the whole system.

Following is an example that shows the percentage of time that the average query spends queued for coordinator activities for each service class, summed across all database partitions:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME,1,19) AS SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,18) AS SUBCLASS_NAME,
       CASE WHEN (SUM(COORD_ACT_COMPLETED_TOTAL) = 0) THEN
         0
       ELSE
         SUM(COORD_ACT_QUEUE_TIME_AVG * COORD_ACT_COMPLETED_TOTAL) /
         SUM(COORD_ACT_COMPLETED_TOTAL)
       END AS AVG_QUEUE_TIME,
       CASE WHEN (SUM(COORD_ACT_COMPLETED_TOTAL) = 0) THEN
         0
       ELSE
         SUM(COORD_ACT_LIFETIME_AVG * COORD_ACT_COMPLETED_TOTAL) /
         SUM(COORD_ACT_COMPLETED_TOTAL)
       END AS AVG_LIFE_TIME,
       CASE WHEN (SUM(COORD_ACT_COMPLETED_TOTAL) = 0) THEN
         0
       ELSE CASE WHEN
         (CAST(SUM(COORD_ACT_LIFETIME_AVG * COORD_ACT_COMPLETED_TOTAL) /
                     SUM(COORD_ACT_COMPLETED_TOTAL) AS INTEGER) = 0) THEN
                      0
                    ELSE
                      100 * (SUM(COORD_ACT_QUEUE_TIME_AVG * COORD_ACT_COMPLETED_TOTAL) /
                                SUM(COORD_ACT_COMPLETED_TOTAL)) /
                               (SUM(COORD_ACT_LIFETIME_AVG * COORD_ACT_COMPLETED_TOTAL) /
                                SUM(COORD_ACT_COMPLETED_TOTAL))
                    END
                  END AS PERCENT_TIME_QUEUED
           FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '', -2)) AS STATS
           GROUP BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
           ORDER BY SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
```

```
SUPERCLASS_NAME     SUBCLASS_NAME      AVG_QUEUE_TIME           AVG_LIFE_TIME            PERCENT_TIME_QUEUED
------------------- ------------------ ------------------------ ------------------------ ------------------------
SYSDEFAULTMAINTENAN SYSDEFAULTSUBCLASS +0.00000000000000E+000   +0.00000000000000E+000   +0.00000000000000E+000
SYSDEFAULTSYSTEMCLA SYSDEFAULTSUBCLASS +0.00000000000000E+000   +0.00000000000000E+000   +0.00000000000000E+000
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS +2.32860100000000E+005   +8.23421424000000E+005   +2.82800000000000E-001
```

The results show that the percentage of time that the average activity spends in the queue is about 28%. If previous experience with the system and workload indicates that this is too high or too low, making adjustments to your thresholds can have an impact on the percentage of time spent queuing.

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: Computing averages and a standard deviation from histograms in a 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 workload management configuration, statistics are collected on each database partition. The following example shows how to obtain the average activity lifetime for a single database partition.

Suppose that you have a single-partition 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.

```
Bin 1 - 0 to 2 seconds
Bin 2 - 2 to 4 seconds
Bin 3 - 4 to 8 seconds
Bin 4 - 8 to 16 seconds
Bin 5 - 16 to 32 seconds
Bin 6 - 32 to 64 seconds
Bin 7 - 64 to 128 seconds
Bin 8 - 128 seconds to infinity
```

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:

```
Bin 1 average lifetime = (0+2)/2 = 1
Bin 2 average lifetime = (2+4)/2 = 3
Bin 3 average lifetime = (4+8)/2 = 6
Bin 4 average lifetime = (8+16)/2 = 12
Bin 5 average lifetime = (16+32)/2 = 24
Bin 6 average lifetime = (32+64)/2 = 48
Bin 7 average lifetime = (64+128)/2 = 96
```

Assume that the following histogram was collected during the measurement period:

```
Bin 1    Bin 2    Bin 3   Bin 4    Bin 5    Bin 6    Bin 7    Bin 8
count    count    count   count    count    count    count    count
   20       30       80      10        5        3        2        0
```

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 database partition 1 as follows:

```
average lifetime = (20 x 1 + 30 x 3 + 80 x 6 + 10 x 12 + 5 x 24 + 3 x 48 + 2 x 96) / 150
                 = (20 + 90 + 480 + 120 + 120 + 144 + 192) / 150
                 = 1166 / 150
                 = 7.77 seconds
```

You can approximate the lifetime standard deviation as follows:

```
Standard deviation = [(20 x (1 - 7.77)2 + 30 x (3 - 7.77)2 + ... ) / 150]1/2
```

For partitioned database environments, averages and standard deviations can be computed by first computing a combined histogram across all database partitions by adding the counts of each bin across the database partitions.

For example, assume that the database has two partitions, the histogram bin sizes are as described above, and the histogram has the following data:

```
Database  Bin 1     Bin 2     Bin 3    Bin 4     Bin 5      Bin 6     Bin 7     Bin 8
partition count     count     count    count     count      count     count     count
        1    20        30        80       10         5          3         2         0
        2     1         5        20       20         4          0         0         0
```

Because the bin sizes are the same across all database partitions, the overall histogram is easy to compute:

```
Bin 1     Bin 2     Bin 3     Bin 4     Bin 5     Bin 6     Bin 7     Bin 8
count     count     count     count     count     count     count     count
   21        35       100        30         9         3         2         0
```

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-partition environment:

```
Average lifetime = (21 x 1 + 35 x 3 + 100 x 6 + 30 x 12 + 9 x 24 + 3 x 48 + 2 x 96) / 200
                 = (21 + 105 + 600 + 360 + 216 + 144 + 192) / 200
                 = 1638 / 200
                 = 8.19 seconds
```

$$\text{Standard deviation} = [(21 \times (1 - 8.19)^2 + 35 \times (3 - 7.77)^2 + ...) / 200]^{1/2}$$

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 partitions by using the data obtained from the table function WLM_GET_SERVICE_CLASS_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 partitions. 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_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 SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME

SUPERCLASS_NAME     SUBCLASS_NAME      ACTSCOMPLETED ACTSABORTED ACTSHW ACTAVGLIFETIME
------------------- ------------------ ------------- ----------- ------ --------------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 8             0           1               3.750
BI_APPS             SYSDEFAULTSUBCLASS 4             0           1              14.230
BATCH               SYSDEFAULTSUBCLASS 1             0           1              25.600
```

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

```
SUPERCLASS_NAME     SUBCLASS_NAME      ACTSCOMPLETED ACTSABORTED ACTSHW ACTAVGLIFETIME
------------------- ------------------ ------------- ----------- ------ --------------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 8             0           1               3.750
BI_APPS             SYSDEFAULTSUBCLASS 4             0           1               5.230
BATCH               SYSDEFAULTSUBCLASS 1             0           1              25.600
```

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

```
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 AVGLIFETIME
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('', '', -2)) AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME

SUPERCLASS_NAME     SUBCLASS_NAME      PART AVGLIFETIME
------------------- ------------------ ---- -----------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0          3.425
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1          2.752
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2          8.230
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3          0.593
```

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

Many different situations can cause a system slowdown. Use the following principles to make the best use of the information provided by the 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, 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 CPU, I/O bandwidth, and other 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 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 partitions 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 (a wildcard character) to indicate that data is to be gathered for all service classes on all database partitions.

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_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
```

```
SUPERCLASS_NAME     SUBCLASS_NAME      ACTSCOMPLETED ACTSABORTED ACTSHW ACTAVGLIFETIME
------------------- ------------------ ------------- ----------- ------ --------------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 20            0           1               3.750
SUP1                SUB1               40            0           8               7.223
```

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:

```
SELECT SUBSTR(WLSTATS.WORKLOAD_NAME,1,22) AS WL_NAME,
       SUBSTR(CHAR(WLSTATS.DBPARTITIONNUM),1,4) AS PART,
       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, PART;

WL_NAME                PART WLO_HIGH_WTRMRK WLO_ACT_HIGH_WTRMRK
---------------------- ---- --------------- -------------------
LYNNSALES                 0               2                   8
LYNNSALES                 1               0                   0
SYSDEFAULTUSERWORKLOAD    0               1                   1
SYSDEFAULTUSERWORKLOAD    1               0                   0
```

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.

Example: Analyzing workloads by activity type

You can use 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:

```
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 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 (a work action must also exist for each work class; otherwise activities are not classified in the work class). The * represents all activities that do not fall into the READCLASS or LOADCLASS work class.

```
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       2005-11-25-18.52.50.478000 0
AdminActionSet       0    LoadClass       2005-11-25-18.52.49.343000 2
AdminActionSet       1    LoadClass       2005-11-25-18.52.50.478000 0
AdminActionSet       0    *               2005-11-25-18.52.50.478000 0
AdminActionSet       1    *               2005-11-25-18.52.50.478000 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_CLASS_STATS table function:

```
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 AVGLIFETIME
FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS('MYSUPERCLASS', 'LOADSERVICECLASS', -2))
AS SCSTATS
ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART

SUPERCLASS_NAME     SUBCLASS_NAME      PART AVGLIFETIME
------------------- ------------------ ---- ----------------------
SYSDEFAULTUSERCLASS LOADSERVICECLASS   0                  4691.242
SYSDEFAULTUSERCLASS LOADSERVICECLASS   1                  4644.740
SYSDEFAULTUSERCLASS LOADSERVICECLASS   2                  4612.431
SYSDEFAULTUSERCLASS LOADSERVICECLASS   3                  4593.451
```

Example: Identifying hung activities

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 a hung activity

Following is an example of identifying a hung query. Assume that a user from the Sales department who is running the SalesReport application complains that the application is hung.

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:

```
SELECT SUBSTR(CHAR(COORD_PARTITION_NUM),1,5) AS COORD,
SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
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 PART, UOWID, ACTID

COORD PART 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:

```
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_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:

```
APPLICATION_HANDLE UOW_ID ACTIVITY_ID REQUEST_TYPE EVENT_TYPE EVENT_OBJECT
------------------ ------ ----------- ------------ ---------- ------------
                 1      2           5         OPEN       WAIT    WLM_QUEUE
```

If the activity were executing, the result would be:

```
APPLICATION_HANDLE UOW_ID ACTIVITY_ID REQUEST_TYPE EVENT_TYPE EVENT_OBJECT
------------------ ------ ----------- ------------ ---------- ------------
                 1      2           5          OPEN   PROCESS      REQUEST
```

If the activity were waiting on a lock, the result would be:

```
APPLICATION_HANDLE UOW_ID ACTIVITY_ID REQUEST_TYPE EVENT_TYPE EVENT_OBJECT
------------------ ------ ----------- ------------ ---------- ------------
                 1      2           5         OPEN    ACQUIRE         LOCK
```

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 hung 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 allows you to capture information about a specific query, as opposed to every statement running at the time. You can also obtain the statement text, truncated to the first 1024 characters, by using WLM_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 a hung application. Also assume that you have either the application name or the authorization ID of the hung 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:

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
       SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
       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, PART, AGENT_TID

APPHANDLE PART AGENT_TID AGENTTYPE   EVENTOBJECT  REQTYPE UOW_ID ACT_ID
```

```
--------- ---- --------- ----------- -----------  ------- ------ ------
2         0    1         COORDINATOR REQUEST      OPEN    2      1
2         1    3         SUBAGENT    LOCK         -       2      1
```

The results indicate that agent 1 is waiting on a remote response. Looking at the agent on the remote partition 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:

```
SELECT AGENT_ID AS WAITING_FOR_LOCK,
       SUBSTR(APPL_ID_HOLDING_LK,1,40) AS HOLDING_LOCK,
       CAST(LOCK_MODE_REQUESTED AS SMALLINT) AS WANTED,
       CAST(LOCK_MODE AS SMALLINT) AS HELD
FROM TABLE(SNAPSHOT_LOCKWAIT('SAMPLE',-1)) AS SLW

WAITING_FOR_LOCK     HOLDING_LOCK                             WANTED HELD
-------------------- ---------------------------------------- ------ ------
                   2 *LOCAL.DB2.060131021547                       9      5
```

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 hung activity, you can use the WLM_CANCEL_ACTIVITY procedure. If the successful completion of the hung application is more important than the successful completion of the lock-owning application, you can force the lock-owning application.

Example: Capturing information about an activity for later analysis

You can use workload management tools to capture information about an activity for later analysis.

Assume that you have a stored procedure called MYSCHEMA.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 MYSCHEMA.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 workload management objects:

```
CREATE SERVICE CLASS SC1;
CREATE WORKLOAD WL1 APPLNAME ('DB2BP') SERVICE CLASS SC1;
CREATE SERVICE CLASS PROBLEMQUERIESSC UNDER SC1 COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS;

CREATE WORK CLASS SET PROBLEMQUERIES
(WORK CLASS CALLSTATEMENTS WORK TYPE CALL ROUTINES IN SCHEMA MYSCHEMA);

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 PROBLEMQUERIESSC);
```

After the MYSCHEMA.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:

```
AGENT_ID             UOW_ID      ACTIVITY_ID
==================== =========== ===========
                   1           2           3
```

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
         CHILD.APPL_ID = CHILDSTMT.APPL_ID AND
         PARENT.UOW_ID = CHILD.PARENT_UOW_ID AND
         CHILD.UOW_ID = CHILDSTMT.UOW_ID AND
         PARENT.ACTIVITY_ID = CHILD.PARENT_ACTIVITY_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:

```
UOW_ID ACTIVITY_ID STMT_TEXT                           LIFE_TIME
====== =========== =================================== =============
2      3           CALL SLOWPROC                                1000
2      4           SELECT COUNT(*) FROM ORG                        1
2      5           SELECT * FROM MYHUGETABLE                     999
```

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.

Example: Investigating agent usage by service class

The workload management solution 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:

```
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

SUPERCLASS_NAME     SUBCLASS_NAME       AGENT_COUNT
------------------- ------------------- -----------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS            7
TEST                SYSDEFAULTSUBCLASS           20
```

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 allowed for a workload or a service class. Alternatively, you can restrict the number of connections for a service class.

Example: Tuning a 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 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 41. Results of capacity planning

Type of work	Application	Goal	Importance	Expected throughput
Order entry	`orderentryapp.exe`	Obtain an average response time < 1 second	High	10 000 (both inserts and updates) per day

Table 41. Results of capacity planning (continued)

Type of work	Application	Goal	Importance	Expected throughput
Business intelligence queries	`businessobjects.exe`	Obtain an average response time < 10 seconds	High	100 queries per day
Batch processing	`batchapp.exe`	Maximize throughput	Low	5000 updates per day
Other	All other applications	Best effort	Low	100 activities per day

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 42. Activities each day

Service class	Predicted activities per day	Actual activities per day
ORDER_ENTRY_SC	10 000	9700
BI_QUERIES_SC	100	115
BATCH_SC	5000	5412
SYSDEFAULTUSERCLASS	100	85

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 43. Response times

Service class	Response time goal	Actual average lifetime
ORDER_ENTRY_SC	< 1 second	0.8 seconds
BI_QUERIES_SC	< 10 seconds	30 seconds
BATCH_SC		2 seconds
SYSDEFAULTUSERCLASS		10 minutes

Using the workload management interfaces, 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 CPU 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 CPU 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 CPU 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 CPU reallocation, activities in the SYSDEFAULTUSERCLASS service class have had their response times doubled.

Table 44. Response times after reconfiguration

Service class	Response time goal	Actual average lifetime
ORDER_ENTRY_SC	< 1 second	0.6 seconds
BI_QUERIES_SC	< 10 seconds	9.5 seconds
BATCH_SC		1.5 seconds
SYSDEFAULTUSERCLASS		20 minutes

Example: Tuning a workload management configuration when capacity planning information is unavailable

You can use the 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 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 in-memory 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 is not partitioned.

```
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, 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 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;


TOP_IN_MINUTES   PERCENT_IN_BIN
---------------- --------------
           0.000           0.00
           0.000           0.00
           0.000           0.00
           0.000           0.00
           0.000           0.00
           0.000           0.00
           0.000           0.00
           0.000           0.00
           0.000           0.00
           0.001           0.00
           0.001           0.00
           0.002           0.00
           0.004           0.00
           0.006           0.00
           0.009           0.00
           0.014           0.00
           0.021           0.00
           0.033           0.00
           0.050           0.00
           0.077           0.00
           0.118           0.00
           0.180           0.00
           0.274           0.00
           0.419           0.00
           0.639           0.00
           0.975           0.00
           1.488           0.00
           2.269           0.00
           3.462           0.00
           5.280           0.00
           8.054           0.00
          12.286           0.00
          18.740           0.00
          28.584          10.00
          43.600          15.00
          66.505          45.00
         101.442          23.00
         154.731           5.00
         236.015           2.00
         360.000           0.00
```

The following figure shows the results of the preceding query plotted as a graph:

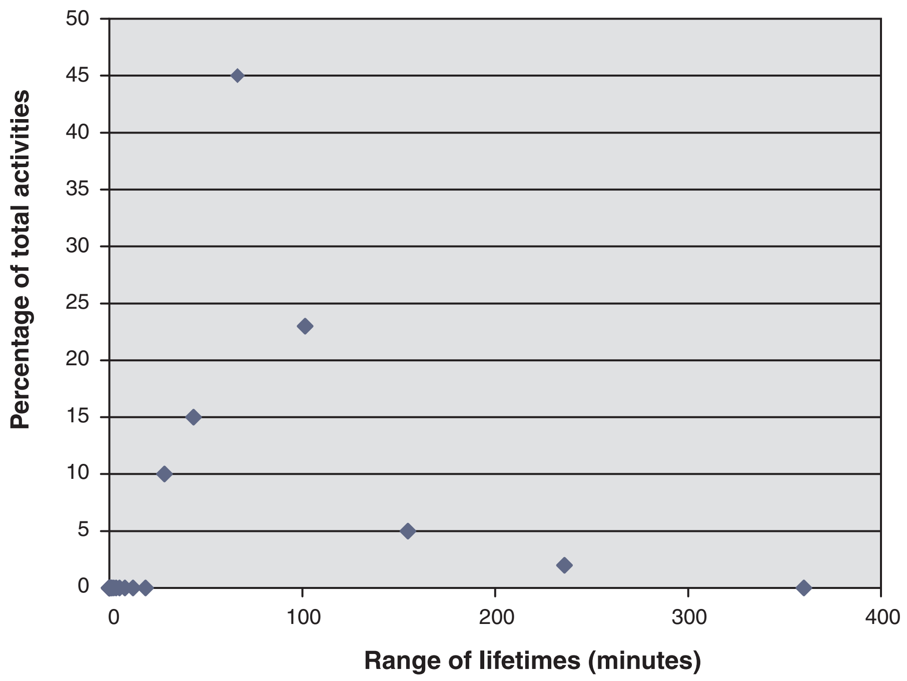


Figure 27. Activity lifetime histogram of unclassified activities

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 allow the data collection to 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.

Example: 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.

Part 5. Reference

Chapter 8. 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

```
►►—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

EXECUTE privilege on the WLM_CANCEL_ACTIVITY procedure.

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 for the activity.

WLM_CAPTURE_ACTIVITY_IN_PROGRESS - Collect activity information for activities event monitor

This procedure causes information on a given activity to be gathered and written to the active activities event monitor. When applied to an activity that has child activities, this procedure recursively generates a record for each child activity all the way down to the lowest level. This information is collected and sent at the instant this procedure is called. It does not wait until the activity completes execution. The record of the activity in the event monitor is marked as a partial record.

Syntax

```
►►──WLM_CAPTURE_ACTIVITY_IN_PROGRESS──(──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 captured. 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 captured. 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 captured. If the argument is null, no activity will be found and an SQL4702N with SQLSTATE 5U035 is returned.

Authorization

EXECUTE privilege on the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure.

Example

A particular procedure MYSCHEMA.MYSLOWSTP might be running more slowly than usual. A user complains and the administrator wants to investigate the cause of the slowdown. Investigating while the stored procedure is executing can be impractical, so the administrator has the ability to capture the stored procedure activity and any of the activities nested within it.

Assuming that an event monitor for DB2 activities named DB2ACTIVITIES exists and has been activated, the administrator can use the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES 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 WLM_CAPTURE_ACTIVITY_IN_PROGRESS as follows:

```
CALL WLM_CAPTURE_ACTIVITY_IN_PROGRESS(1,2,3)
```

Once the procedure has completed, for an activity event monitor named DB2ACTIVITIES, the administrator can use the following table function to find out where the activity spent its time:

```
 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, CHILDSTMT.STMT_TEXT, CHILD.ACT_EXEC_TIME
   FROM RAH PARENT, ACTIVITY_DB2ACTIVITIES CHILD,
        ACTIVITYSTMT_DB2ACTIVITIES CHILDSTMT
   WHERE PARENT.APPL_ID = CHILD.APPL_ID AND
         CHILD.APPL_ID = CHILDSTMT.APPL_ID AND
         PARENT.UOW_ID = CHILD.PARENT_UOW_ID AND
         CHILD.UOW_ID = CHILDSTMT.UOW_ID AND
         PARENT.ACTIVITY_ID = CHILD.PARENT_ACTIVITY_ID AND
         CHILD.ACTIVITY_ID = CHILDSTMT.ACTIVITY_ID AND
         PARENT.LEVEL < 64
  )
SELECT UOW_ID, ACTIVITY_ID, SUBSTR(STMT_TEXT,1,40),
                ACT_EXEC_TIME AS
                LIFE_TIME
FROM RAH
```

An example of a query to use the table function is:

```
SELECT * FROM TABLE(SHOWCAPTUREDACTIVITY(1, 2, 3))
  AS ACTS ORDER BY UOW_ID, ACTIVITY_ID
```

Usage notes

If there is no active activities event monitor, an SQL1633W with SQLSTATE 01H53 is returned.

If you are using this procedure to collect activity information, input data values will not be collected.

WLM_COLLECT_STATS - Collect and reset workload management statistics

This procedure causes statistics for service classes, workloads, work classes and threshold queues to be gathered and written to the statistics event monitor. The statistics for service classes, workloads, work classes and threshold queues are also reset. If there is no active statistics event monitor, then the statistics are only reset.

Syntax

```
►►──WLM_COLLECT_STATS──(──)────────────────────────────────────────────────►◄
```

The schema is SYSPROC.

Authorization

EXECUTE privilege on the WLM_COLLECT_STATS procedure.

Examples

Example 1: Call WLM_COLLECT_STATS to collect and reset statistics.

```
CALL WLM_COLLECT_STATS()
```

The following is an example of output from this query.

```
Return Status = 0
```

Example 2: Call WLM_COLLECT_STATS to collect and reset statistics while another call is in progress.

```
CALL WLM_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.
```

Usage notes

The WLM_COLLECT_STATS procedure is used to manually collect statistics. It performs the same collect (send statistics to the active statistics event monitor) and reset operations that occur automatically on the interval defined by the WLM_COLLECT_INT database configuration parameter. If the procedure is invoked at the same time as another collect and reset request is in progress (for example, the procedure is invoked at same time as another invocation of the procedure is running, or at the same time an automated collection occurs) a warning, SQL1632W with SQLSTATE 01H53 is returned and the request is ignored.

The WLM_COLLECT_STATS procedure only starts the collection and reset process. It might return before the process has completed, that is, 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 occurs, the call to the WLM_COLLECT_STATS procedure (which is itself an activity and will be counted in activity statistics) might be counted in either the prior collection interval or the new collection interval that has just started.

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.

Syntax

```
►►──WLM_GET_ACTIVITY_DETAILS──(──application_handle──,──uow_id──,──────────────────►

►─activity_id──,──dbpartitionnum──)──────────────────────────────────────────────►◄
```

The schema is SYSPROC.

Table function parameters

application_handle
An input argument of type BIGINT that specifies a valid application handle. If the argument is null, no rows are returned from this function. If the argument is null, an SQL171N 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. If the argument is null, an SQL171N 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. If the argument is null, an SQL171N error is returned.

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

Authorization

EXECUTE privilege on the WLM_GET_ACTIVITY_DETAILS function.

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 partition. This example is restricted to showing only an eleven member subset of the name-value pairs for each partition 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 46 on page 187 and Table 47 on page 189.

```
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.

```
PART NAME                 VALUE
---- -------------------- ------------------------------
0    APPLICATION_HANDLE   1
0    COORD_PARTITION_NUM  0
0    LOCAL_START_TIME     2005-11-25-18.52.49.343000
0    UOW_ID               1
0    ACTIVITY_ID          5
0    PARENT_UOW_ID        1
0    PARENT_ACTIVITY_ID   3
0    ACTIVITY_TYPE        READ_DML
0    NESTING_LEVEL        0
0    INVOCATION_ID        1
0    ROUTINE_ID           0
1    APPLICATION_HANDLE   1
1    COORD_PARTITION_NUM  0
1    LOCAL_START_TIME     2005-11-25-18.52.49.598000
1    UOW_ID               1
1    ACTIVITY_ID          5
1    PARENT_UOW_ID
1    PARENT_ACTIVITY_ID
1    ACTIVITY_TYPE        READ_DML
1    NESTING_LEVEL        0
1    INVOCATION_ID        1
1    ROUTINE_ID           0
```

Usage note

An ACTIVITY_STATE of QUEUED means that the coordinator activity has made a RPC to the catalog partition 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 partition has a value of WLM_QUEUE.

Information returned

Table 45. Information returned for WLM_GET_ACTIVITY_DETAILS

Column Name	Data Type	Description
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
NAME	VARCHAR(256)	Element name. See Table 46 on page 187 and Table 47 on page 189 for possible values.
VALUE	VARCHAR(1024)	Element values. See Table 46 on page 187 and Table 47 on page 189 for possible values.

Table 46. Elements returned

Element Name	Description
APPLICATION_HANDLE	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier will be the same on every partition where the application may make a secondary connection.
COORD_PARTITION_NUM	The coordinator partition of the activity.
UOW_ID	Unique unit of work identifier within an application. Refers to the original unit of work this activity started in.
ACTIVITY_ID	Unique activity identifier within an application.
PARENT_UOW_ID	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 partition.
PARENT_ACTIVITY_ID	Unique activity identifier within a unit of work for the parent of the activity whose ID is ACTIVITY_ID. Returns an empty string if the activity has no parent activity.
ACTIVITY_STATE	Possible values include: • CANCEL_PENDING • EXECUTING • IDLE • INITIALIZING • QP_CANCEL_PENDING • QP_QUEUED • QUEUED • TERMINATING • UNKNOWN
ACTIVITY_TYPE	Possible values include: • CALL • DDL • LOAD • OTHER • READ_DML • WRITE_DML
NESTING_LEVEL	This represents the nesting level of this activity. Nesting level is the depth to which this activity is nested within its top-most parent activity.
INVOCATION_ID	This distinguishes one particular invocation of this activity from others at the same nesting level. Returns zero if the activity is not nested.
ROUTINE_ID	Routine unique identifier. Returns zero if the activity is not part of a routine.
UTILITY_ID	If the activity is a utility, this is its utility ID. Otherwise, this field is 0.
SERVICE_CLASS_ID	Unique identifier of the service class to which this activity belongs.

Table 46. Elements returned (continued)

Element Name	**Description**
DATABASE_WORK_ACTION_SET_ID	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.
DATABASE_WORK_CLASS_ID	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.
SERVICE_CLASS_WORK_ACTION_SET_ID	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.
SERVICE_CLASS_WORK_CLASS_ID	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.
ENTRY_TIME	The time that this activity arrived into the system.
LOCAL_START_TIME	The time that this activity began doing work on the partition. It is in local time. This field can be an empty string when an activity has entered the system but is in a queue and has not started executing.
LAST_REFERENCE_TIME	Every time a request occurs in this activity, this field is updated.
PACKAGE_NAME	If the activity is a SQL statement, this represents the name of its package.
PACKAGE_SCHEMA	If the activity is a SQL statement, this represents the schema name of its package.
PACKAGE_VERSION_ID	If the activity is a SQL statement, this represents the version of its package.
SECTION_NUMBER	If the activity is a SQL statement, this represents its section number.
STMT_PKG_CACHE_ID	Statement package cache identifier.
STMT_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. It is an empty string otherwise.
EFFECTIVE_ISOLATION	The effective isolation level for this activity.
EFFECTIVE_LOCK_TIMEOUT	The effective lock timeout value for this activity.
EFFECTIVE_QUERY_DEGREE	The effective value of query degree for this activity.
QUERY_COST_ESTIMATE	Estimated cost, in timerons, for a query, as determined by the SQL compiler.

Table 46. Elements returned (continued)

Element Name	Description
ROWS_FETCHED	This is the number of rows read from the table. This reports only those values for the database partition for which this record is recorded. On DPF systems, these values may not reflect the correct totals for the whole activity. When the statement monitor switch is not turned on, this element is not collected and -1 is written instead.
ROWS_MODIFIED	This is the number of rows inserted, updated, or deleted. This reports only those values for the database partition for which this record is recorded. On DPF systems, these values may not reflect the correct totals for the whole activity. When the statement monitor switch is not turned on, this element is not collected and -1 is written instead.
SYSTEM_CPU_TIME	The total system CPU time (in seconds and microseconds) used by the database manager agent process, the unit of work, or the statement. When either the statement monitor switch or the timestamp switch is not turned on, this element is not collected and -1 is written instead.
USER_CPU_TIME	The total user CPU time (in seconds and microseconds) used by the database manager agent process, the unit of work, or the statement. When either the statement monitor switch or the timestamp switch is not turned on, this element is not collected and -1 is written instead.
QP_QUERY_ID	The query ID assigned to this activity by Query Patroller if the activity is a query. A query ID of 0 indicates that Query Patroller did not assign a query ID to this activity.

The following are returned only if the corresponding thresholds apply to the activity.

Table 47. Elements returned if applicable

Element Name	Description
CONCURRENTWORKLOADACTIVITIES_THRESHOLD_ID	The ID of the threshold.
CONCURRENTWORKLOADACTIVITIES_THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
CONCURRENTWORKLOADACTIVITIES_THRESHOLD_VIOLATED	Yes indicates that this activity violated the threshold. No indicates that this activity has not violated the threshold.
CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_ID	The ID of the threshold.
CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_QUEUED	Whether the activity was queued by this threshold.
CONCURRENTDBCOORDACTIVITIES_DB_THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_ID	The ID of the threshold.

Table 47. Elements returned if applicable (continued)

Element Name	**Description**
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_QUEUED	'Yes' indicates that the activity was queued by this threshold. 'No' indicates that the activity was not queued.
CONCURRENTDBCOORDACTIVITIES_WORK_ACTION_SET _THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_ID	The ID of the threshold.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_QUEUED	'Yes' indicates that the activity was queued by this threshold. 'No' indicates that the activity was not queued.
CONCURRENTDBCOORDACTIVITIES_SUPERCLASS _THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_ID	The ID of the threshold.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_QUEUED	Whether the activity was queued by this threshold.
CONCURRENTDBCOORDACTIVITIES_SUBCLASS _THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.
ESTIMATEDSQLCOST_THRESHOLD_ID	The ID of the threshold.
ESTIMATEDSQLCOST_THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
ESTIMATEDSQLCOST_THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.
SQLTEMPSPACE_THRESHOLD_ID	The ID of the threshold.
SQLTEMPSPACE_THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
SQLTEMPSPACE_THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.
SQLROWSRETURNED_THRESHOLD_ID	The ID of the threshold.
SQLROWSRETURNED_THRESHOLD_VALUE	The value that, when exceeded, will trigger the threshold.
SQLROWSRETURNED_THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.
ACTIVITYTOTALTIME_THRESHOLD_ID	The ID of the threshold.

Table 47. Elements returned if applicable (continued)

Element Name	Description
ACTIVITYTOTALTIME_THRESHOLD_VALUE	A timestamp that is computed by adding the ACTIVITYTOTALTIME threshold duration to the activity entry time. If the activity is still executing when this timestamp is reached, the threshold will be violated.
ACTIVITYTOTALTIME_THRESHOLD_VIOLATED	'Yes' indicates that the threshold has been violated. 'No' indicates that the threshold has not yet been violated.

WLM_GET_QUEUE_STATS table function - Return threshold queue statistics

This function returns basic statistics for one or more threshold queues.

This function returns one row of statistics for each threshold queue. Statistics are returned for queues on all active partitions.

Syntax

```
►►──WLM_GET_QUEUE_STATS──(──threshold_predicate──,──threshold_domain──,──────────►

►─threshold_name──,──threshold_id──)──────────────────────────────────────────►◄
```

The schema is SYSPROC.

Table function parameters

threshold_predicate
: An input argument of type VARCHAR(27) that specifies a valid threshold predicate. The possible values are:

- CONCDBC: concurrent database coordinator activities threshold
- DBCONN: total database partition connections threshold
- SCCONN: total service class partition connections threshold
- NULL or an empty string: data is returned for all possible threshold predicates. 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 valid threshold domain. The possible values are:

- DB: database
- SB: service subclass
- SP: service superclass
- WA: work action set
- NULL or an empty string: data is returned for all possible threshold domains. 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 valid 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 valid 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.

Authorization

EXECUTE privilege on the WLM_GET_QUEUE_STATS function.

Example

To see all the basic statistics for all the queues on the system, across all partitions:

```
  SELECT substr(THRESHOLD_NAME, 1, 6) THRESHNAME,
       THRESHOLD_PREDICATE,
       THRESHOLD_DOMAIN,
       DBPARTITIONNUM PART,
       QUEUE_SIZE_TOP,
       QUEUE_TIME_TOTAL,
       QUEUE_ASSIGNMENTS_TOTAL QUEUE_ASSIGN
  FROM table(WLM_GET_QUEUE_STATS('', '', '', -1)) as QSTATS
```

The following is an example of output from this query.

```
THRESHNAME THRESHOLD_PREDICATE        THRESHOLD_DOMAIN   ...
---------- -------------------------- ------------------ ...
LIMIT1     CONCDBC                    DB                 ...
LIMIT2     SCCONN                     SP                 ...
LIMIT3     DBCONN                     DB                 ...
```

Output from this query (continued).

```
... PART QUEUE_SIZE_TOP QUEUE_TIME_TOTAL QUEUE_ASSIGN
... ---- -------------- ---------------- ------------
... 0                12          1238540          734
... 0                 4           741249           24
... 0                 7           412785          128
```

Usage note

No aggregation across queues (on a partition), or across partitions (for a queue or more) is performed, however this type of aggregation can be achieved using SQL queries as shown in the example above.

Information returned

Table 48. Information returned for WLM_GET_QUEUE_STATS

Column Name	Data Type	Description
THRESHOLD_PREDICATE	VARCHAR(27)	Threshold predicate of the threshold responsible for this queue. The possible values are: • CONCDBC: concurrent database coordinator activities threshold • DBCONN: total database partition connections threshold • SCCONN: total service class partition connections threshold The threshold predicate values match those of the THRESHOLDPREDICATE column in the SYSCAT.THRESHOLDS view.
THRESHOLD_DOMAIN	VARCHAR(18)	Domain of the threshold responsible for this queue. The possible values are: • DB: database • SB: service subclass • SP: service superclass • WA: work action set The threshold domain values match those of the DOMAIN column in the SYSCAT.THRESHOLDS view.
THRESHOLD_NAME	VARCHAR(128)	The unique name of the threshold responsible for this queue. The threshold name value matches that of the THRESHOLDNAME column in the SYSCAT.THRESHOLDS view.
THRESHOLD_ID	INTEGER	The unique ID of the threshold responsible for this queue. The threshold ID value matches that of the THRESHOLDID column in the SYSCAT.THRESHOLDS view.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass that is the domain for the threshold responsible for this queue. Null if the domain of the threshold is not a service superclass or service subclass.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass that is the domain for the threshold responsible for this queue. Null if the domain of the threshold is not a service subclass.
WORK_ACTION_SET_NAME	VARCHAR(128)	Name of the work action set that is the domain for the threshold responsible for this queue. Null if the domain of the threshold is not a work action set.

Table 48. Information returned for WLM_GET_QUEUE_STATS (continued)

Column Name	Data Type	Description
WORK_CLASS_NAME	VARCHAR(128)	Name of the work class whose work action belongs to the work action set that is the domain for the threshold responsible for this queue. Null if the domain of the threshold is not a work action set.
WORKLOAD_NAME	VARCHAR(128)	Name of the workload that is the domain for the threshold responsible for this queue. Null if the domain of the threshold is not a workload.
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that can occur that will trigger a reset of statistics, which will update this timestamp: • The WLM_COLLECT_STATS procedure is called. • The periodic collection and reset process controlled by the WLM_COLLECT_INT configuration parameter causes a collection and reset. • The database is reactivated. • The threshold for which queue statistics are being reported was modified and the change was committed. The LAST_RESET timestamp is in local time.
QUEUE_SIZE_TOP	INTEGER	Highest number of connections or activities in the queue that has been reached since the last reset.
QUEUE_TIME_TOTAL	BIGINT	Sum of the times spent in the queue for all connections or activities placed in this queue since the last reset. Units are milliseconds.
QUEUE_ASSIGNMENTS_TOTAL	BIGINT	Number of connections or activities that were assigned to this queue since the last reset.
QUEUE_SIZE_CURRENT	INTEGER	Number of connections or activities in the queue.
QUEUE_TIME_LATEST	BIGINT	Time spent in the queue by the last connection or activity to leave the queue. This is measured in milliseconds.
QUEUE_EXIT_TIME_LATEST	TIMESTAMP	Time that the last connection or activity left the queue.
THRESHOLD_CURRENT_CONCURRENCY	INTEGER	Number of connections or activities that are currently executing according to the threshold.

Table 48. Information returned for WLM_GET_QUEUE_STATS (continued)

Column Name	Data Type	Description
THRESHOLD_MAX_CONCURRENCY	INTEGER	Maximum number of connections or activities that the threshold allows to be concurrently executing.

WLM_GET_SERVICE_CLASS_AGENTS - List agents executing in a service class

This function returns the list of agents, fenced mode processes (db2fmps) and system entities on the given partition that are executing in the given service class or on behalf of the given application. The system entities are non-agent threads and processes, such as page cleaners and prefetchers.

Syntax

```
►►──WLM_GET_SERVICE_CLASS_AGENTS──(──service_superclass_name──,──────────────────►

►─service_subclass_name──,──application_handle──,──dbpartitionnum──)────────────►◄
```

The schema is SYSPROC.

Table function parameters

service_superclass_name
: An input argument of type VARCHAR(128) that specifies a valid service superclass name in the same database as the one currently connected to when calling this function. If the argument is null or an empty string, data is retrieved for all the superclasses in the database for which the other parameters match.

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 for which the other parameters match.

application_handle
: An input argument of type BIGINT that specifies the application handle for which agent information should be returned. If the argument is null, data is retrieved for all applications in the database for which the other parameters match. An application handle of 0 will return the system entities only.

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

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_CLASS_AGENTS function.

Example

Return a list of agents that are associated with application handle 1 for all database partitions. The application handle could have been determined using the LIST APPLICATIONS command or the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES table function.

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
     SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
     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(WLM_GET_SERVICE_CLASS_AGENTS(CAST(NULL AS VARCHAR(128)),
     CAST(NULL AS VARCHAR(128)), 1, -2)) AS SCDETAILS
ORDER BY APPHANDLE, PART, AGENT_TID
```

The following is an example of output from this query.

```
APPHANDLE PART AGENT_TID AGENTTYPE   AGENTSTATE REQTYPE       UOW_ID ACT_ID
--------- ---- --------- ----------- ---------- ------------- ------ ------
1         0    3         COORDINATOR ACTIVE     FETCH         1      5
1         0    4         SUBAGENT    ACTIVE     SUBSECTION:1  1      5
1         1    2         SUBAGENT    ACTIVE     SUBSECTION:2  1      5
```

Here we see a coordinator agent and a subagent on partition 0 as well as a subagent on partition 1 operating on behalf of an activity with UOW id 1 and activity id 5. The coordinator agent tells us that the request is a fetch request.

Usage note

The parameters have the effect of being ANDed together. That is, if one were to specify conflicting records such as a service superclass SUP_A and subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows would be returned.

Information returned

Table 49. Information returned by WLM_GET_SERVICE_CLASS_AGENTS

Column Name	Data Type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
APPLICATION_HANDLE	BIGINT	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier will be the same on every partition where the application may make a secondary connection.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
ENTITY	VARCHAR(32)	If the type of entity in this row is an agent, this field shows ″db2agent″. If the type of entity in this row is a fenced mode process, this field shows ″db2fmp (pid)″ where pid is the process ID of the fenced mode process. Otherwise, the name of the system entity is shown.
WORKLOAD_NAME	VARCHAR(128)	Name of the workload from which this record was collected.

Table 49. Information returned by WLM_GET_SERVICE_CLASS_AGENTS (continued)

Column Name	Data Type	Description
WORKLOAD_OCCURRENCE_ID	INTEGER	The ID of the workload occurrence. This does not uniquely identify the workload occurrence unless it is coupled with the coordinator database partition number and the workload name. Alternatively, the application handle can be used instead of the coordinator database partition number.
UOW_ID	INTEGER	Unique unit of work identifier within an application. Refers to the original unit of work this activity started in.
ACTIVITY_ID	INTEGER	Unique activity identifier within a unit of work.
PARENT_UOW_ID	INTEGER	Unique unit of work identifier within an application. Refers to the original unit of work this activity's parent activity started in. Returns null if this activity has no parent.
PARENT_ACTIVITY_ID	INTEGER	Unique activity identifier within a unit of work for the parent of the activity whose ID is activity_id. Returns null if this activity has no parent.
AGENT_TID	BIGINT	Thread ID of the agent or system entity. If this ID is unavailable, this field is null.
AGENT_TYPE	VARCHAR(32)	Coordinator or subagent. If coordinator, the agent ID may change in concentrator environments. The agent types are represented by: • COORDINATOR • OTHER • PDBSUBAGENT • SMPSUBAGENT
SMP_COORDINATOR	INTEGER	Whether or not the agent is an smp coordinator: 1 for yes and 0 for no.
AGENT_SUBTYPE	VARCHAR(32)	Possible subtypes include: • DSS • OTHER • RPC • SMP
AGENT_STATE	VARCHAR(32)	Whether an agent is associated or active. The possible values are: • ACTIVE • ASSOCIATED
EVENT_TYPE	VARCHAR(32)	The type of event last processed by this agent. The possible values are: • ACQUIRE • PROCESS • WAIT

Table 49. Information returned by WLM_GET_SERVICE_CLASS_AGENTS (continued)

Column Name	Data Type	Description
EVENT_OBJECT	VARCHAR(32)	The object of the event last processed by this agent. The possible values are: • COMPRESSION_DICTIONARY_BUILD • IMPLICIT_REBIND • INDEX_RECREATE • LOCK • LOCK_ESCALATION • QP_QUEUE • REMOTE_REQUEST • REQUEST • ROUTINE • WLM_QUEUE
EVENT_STATE	VARCHAR(32)	The state of the event last processed by this agent. The possible values are: • EXECUTING • IDLE
REQUEST_ID	VARCHAR(64)	Unique only in combination with application_handle. This can be used for distinguishing between having one request take a long time versus having multiple requests. For examples, distinguishing multiple fetches from one long fetch.
REQUEST_TYPE	VARCHAR(32)	The type of request. The possible values are: • For coordinator agents: – CLOSE – COMMIT – COMPILE – DESCRIBE – EXCSQLSET – EXECIMMD – EXECUTE – FETCH – INTERNAL <number> – OPEN – PREPARE – REBIND – REDISTRIBUTE – REORG – ROLLBACK – RUNSTATS • For subagents (DSS and SMP): – displays the subsection number in the form ″SUBSECTION:<subsection number>″ if the subsection number is non-zero. Otherwise, returns NULL.

Table 49. Information returned by WLM_GET_SERVICE_CLASS_AGENTS (continued)

Column Name	Data Type	Description
REQUEST_TYPE (continued)	VARCHAR(32)	• For subagents (RPC): – ABP – CATALOG – INTERNAL – REORG – RUNSTATS – WLM • For subagents (OTHER): – ABP – APP_RBSVPT – APP_RELSVPT – BACKUP – CLOSE – EXTERNAL_RBSVPT – EVMON – FORCE – FORCE_ALL – INTERNAL <number> – INTERRUPT – NOOP: if there is no request – QP – REDISTRIBUTE – STMT_RBSVPT – STOP_USING – UPDATE_DBM_CFG – WLM If the request type is one of the internal types, the value is displayed as ’INTERNAL’ followed by the actual value of the internal constant.
NESTING_LEVEL	INTEGER	This represents the nesting level of the activity whose ID is activity_id. Nesting level is the depth to which this activity is nested within its top-most parent activity.
INVOCATION_ID	INTEGER	This distinguishes one particular invocation of an activity from others at the same nesting level.
ROUTINE_ID	INTEGER	Routine unique identifier. Null if not part of a routine.

WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES - List of workload occurrences

This function returns the list of all workload occurrences executing in a given service class on a particular partition. A workload occurrence is a specific database connection whose attributes match with the definition of a workload and hence is associated with or assigned to the workload.

Syntax

```
►►──WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES──(──service_superclass_name──,────────►

►──service_subclass_name──,──dbpartitionnum──)──────────────────────────────────────►◄
```

The schema is SYSPROC.

Table function parameters

service_superclass_name
: An input argument of type VARCHAR(128) that specifies a valid service superclass name 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 for which the other parameters match.

service_subclass_name
: An input argument of type VARCHAR(128) that specifies a valid service superclass name in the currently connected database. If the argument is null or an empty string, the data is retrieved for all the subclasses in the database for which the other parameters match.

dbpartitionnum
: An input argument of type INTEGER that specifies a valid partition number in the same instance as the currently connected database. Indicate -1 for the current database partition, or -2 for all database partitions. If the null value is specified, -1 is set implicitly.

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES function.

Example

If an administrator would like to see what workload occurrences are running on the system as a whole, the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES function can be called with a null value or an empty string for *service_superclass_name* and *service_subclass_name*, and -2 for *dbpartitionnum*.

```
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,
       SUBSTR(CHAR(COORD_PARTITION_NUM),1,4) AS COORDPART,
       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, PART, APPHNDL,
       WORKLOAD_NAME, WLO_ID
```

Assuming that the system has four database partitions and is running two workloads at this time, the above query would produce a result like the following:

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART COORDPART ...
------------------- ------------------ ---- --------- ...
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 2    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    0         ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    0         ...
```

Output from this query (continued).

```
... APPHNDL WORKLOAD_NAME         WLO_ID
... ------- --------------------- ------
... -       -                     -
... -       -                     -
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
... 1       SYSDEFAULTUSERWORKLOAD 1
... 2       SYSDEFAULTUSERWORKLOAD 2
```

Usage note

The parameters have the effect of being ANDed together. That is, if one were to specify conflicting records such as a service superclass SUP_A and subclass SUB_B such that SUB_B is not a subclass of SUP_A, no rows would be 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 50. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES

Column Name	Data Type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
COORD_PARTITION_NUM	SMALLINT	Partition number of the coordinator partition of the given workload occurrence.
APPLICATION_HANDLE	BIGINT	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier will be the same on every partition where the application may make a secondary connection.
WORKLOAD_NAME	VARCHAR(128)	Name of the workload from which this record was collected.

Table 50. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES (continued)

Column Name	Data Type	Description
WORKLOAD_OCCURRENCE_ID	INTEGER	The ID of the workload occurrence. This does not uniquely identify the workload occurrence unless it is coupled with the coordinator database partition number and the workload name. Alternatively, the application handle can be used instead of the coordinator database partition number.
WORKLOAD_OCCURRENCE_STATE	VARCHAR(32)	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 Query Patroller or a workload management queuing threshold. In a database partitioning feature (DPF) environment, this state may indicate that the coordinator agent has made an RPC to the catalog partition 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.
UOW_ID	INTEGER	Unique unit of work identifier within an application. Refers to the original unit of work this workload occurrence started in.
SYSTEM_AUTH_ID	VARCHAR(128)	System authorization ID under which the workload occurrence was injected into the system.
SESSION_AUTH_ID	VARCHAR(128)	Session authorization ID under which the workload occurrence was injected into the system.
APPLICATION_NAME	VARCHAR(128)	The name of the application that created this workload occurrence.
CLIENT_WRKSTNNAME	VARCHAR(255)	The current value of the CLIENT_WRKSTNNAME special register for this workload occurrence.
CLIENT_ACCTNG	VARCHAR(255)	The current value of the CLIENT_ACCTNG special register for this workload occurrence.
CLIENT_USER	VARCHAR(255)	The current value of the CLIENT_USER special register for this workload occurrence.
CLIENT_APPLNAME	VARCHAR(255)	The current value of the CLIENT_APPLNAME special register for this workload occurrence.

Table 50. Information returned for WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES (continued)

Column Name	Data Type	Description
COORD_ACT_COMPLETED_TOTAL	INTEGER	The number of coordinator activities at any nesting level completed so far in the current unit of work of this workload occurrence. This statistic is updated every time an activity in this workload occurrence completes and is reset at the beginning of each unit of work.
COORD_ACT_ABORTED_TOTAL	INTEGER	The number of coordinator activities aborted so far in the current unit of work of this workload occurrence. This statistic is updated every time an activity in this workload occurrence is aborted and is reset at the beginning of each unit of work.
COORD_ACT_REJECTED_TOTAL	INTEGER	The number of coordinator activities rejected so far in the current unit of work of this workload occurrence. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action, or a predictive threshold. This statistic is updated every time an activity in this workload occurrence is rejected and is reset at the beginning of each unit of work.
CONCURRENT_ACT_TOP	INTEGER	Highest number of concurrent activities at any nesting level in either executing (which includes idle and waiting) or queued state that has been reached for this workload occurrence in the current unit of work. This statistic is reset at the beginning of each unit of work.

WLM_GET_SERVICE_SUBCLASS_STATS - Return statistics of service subclasses

This function returns basic statistics of one or more service subclasses.

Syntax

```
►►──WLM_GET_SERVICE_SUBCLASS_STATS──(──service_superclass_name──,────────────────►

►─service_subclass_name──,──dbpartitionnum──)────────────────────────────────────►◄
```

The schema is SYSPROC.

Table function parameters

service_superclass_name
: An input argument of type VARCHAR(128) that specifies a valid service superclass name in the same database as the one currently connected to when calling this function. If the argument is null or an empty string, the data is retrieved for all the superclasses in the database.

service_subclass_name
: An input argument of type VARCHAR(128) that specifies a valid service subclass name in the same database as the one currently connected to when

calling this function. If the argument is null or an empty string, the data is retrieved for all the subclasses in the database.

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

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_SUBCLASS_STATS function.

Examples

Example 1: Since every activity has to be mapped to a DB2 Service Class prior to being executed, the global state of the system can be regularly monitored using the service class statistics table functions and querying all the service classes on all the partitions (note that passing a null value for an argument is saying to not restrict the result by that argument, except for the last argument, dbpartitionnum, where -2 means that data from all database partitions are to be returned). The following statement returns service class statistics such as average activity lifetime and standard deviation in seconds:

```
 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 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(CAST(NULL AS VARCHAR(128)),
        CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
  ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART
```

The following is an example of output from this query.

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART ...
------------------- ------------------ ---- ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2    ...
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3    ...
```

Output from this query (continued).

```
... AVGLIFETIME STDDEVLIFETIME LAST_RESET
... ----------- -------------- ----------------
...     691.242         34.322 2006-07-24-11.44
...     644.740         22.124 2006-07-24-11.44
...     612.431         43.347 2006-07-24-11.44
...     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 partition.

```
  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,
         CONCURRENT_ACT_TOP AS ACTTOP,
         CONCURRENT_WLO_TOP AS CONNTOP
  FROM TABLE(WLM_GET_SERVICE_SUBCLASS_STATS(CAST(NULL AS VARCHAR(128)),
         CAST(NULL AS VARCHAR(128)), -2)) AS SCSTATS
  ORDER BY SUPERCLASS_NAME, SUBCLASS_NAME, PART
```

The following is an example of output from this query.

```
SUPERCLASS_NAME     SUBCLASS_NAME      PART ACTTOP    CONNTOP
------------------- ------------------ ---- --------- ---------
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 0           10         7
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 1            0         0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 2            0         0
SYSDEFAULTUSERCLASS SYSDEFAULTSUBCLASS 3            0         0
```

The output of this table function will give the administrator a good high level view of the ″load″ on each partition for a specific database by checking the average execution times and numbers of activities. Any significant variations of the high level gauges returned by these table functions may indicate a change in the load on the system.

Usage notes

Some statistics will only be returned if the COLLECT AGGREGATE ACTIVITY DATA and COLLECT AGGREGATE REQUEST DATA settings for the corresponding service subclass are set to a value other than NONE.

The WLM_GET_SERVICE_SUBCLASS_STATS table function returns one row of data per service subclass and per partition. No aggregation across service classes (on a partition), or across partitions (for a service class or more) is performed. However, aggregation can be achieved through SQL queries as shown in the examples above.

The parameters have the effect of being ANDed together. That is, if one were to specify conflicting records such as a superclass name SUPA and subclass name SUBB such that SUBB is not a subclass of SUPA, no rows would be returned.

Information returned

Table 51. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS

Column Name	Data Type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
SERVICE_SUBCLASS_NAME	VARCHAR(128)	Name of the service subclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.

Table 51. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS (continued)

Column Name	Data Type	Description
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that can occur that will trigger a reset of statistics, which will update this timestamp: • The WLM_COLLECT_STATS procedure is called. • The periodic collection and reset process controlled by the WLM_COLLECT_INT configuration parameter causes a collection and reset. • The database is reactivated. • The service subclass for which statistics are being reported was modified and the change was committed. The LAST_RESET timestamp is in local time.
COORD_ACT_COMPLETED_TOTAL	BIGINT	The total number of coordinator activities that users have submitted since the last reset and completed successfully. This count is updated as each activity completes.
COORD_ACT_ABORTED_TOTAL	BIGINT	The total number of coordinator activities that users have submitted since the last reset and completed with errors. This count is updated as each activity aborts.
COORD_ACT_REJECTED_TOTAL	BIGINT	The total number of coordinator activities that users have submitted since the last reset and were rejected prior to execution instead of being allowed to execute. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action, or a predictive threshold. This count is updated as each activity gets rejected.
CONCURRENT_ACT_TOP	INTEGER	Highest number of concurrent activities at any nesting level in either executing (which includes idle and waiting) or queued state that has been reached for this service subclass.
COORD_ACT_LIFETIME_TOP	BIGINT	High watermark for coordinator activity lifetime, counted at all nesting levels. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds.
COORD_ACT_LIFETIME_AVG	DOUBLE	Arithmetic mean of lifetime for coordinator activities at nesting level 0 associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds.

Table 51. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS (continued)

Column Name	Data Type	Description
COORD_ACT_LIFETIME_STDDEV	DOUBLE	Standard deviation of lifetime for coordinator activities at nesting level 0 associated with this service subclass since the last reset. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. This standard deviation is computed from the coordinator activity lifetime histogram and may be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin.
COORD_ACT_EXEC_TIME_AVG	DOUBLE	Arithmetic mean of the execution times for coordinator activities at nesting level 0 associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds.
COORD_ACT_EXEC_TIME_STDDEV	DOUBLE	Standard deviation of the execution times for coordinator activities at nesting level 0 associated with this service subclass since the last reset. Units are milliseconds. This standard deviation is computed from the coordinator activity executetime histogram and might be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin.
COORD_ACT_QUEUE_TIME_AVG	DOUBLE	Arithmetic mean of the queue time for coordinator activities at nesting level 0 associated with this service subclass since the last reset. If the internally tracked average has overflowed, the value -2 is returned. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds.
COORD_ACT_QUEUE_TIME_STDDEV	DOUBLE	Standard deviation of the queue time for coordinator activities at nesting level 0 associated with this service subclass since the last reset. Null when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds. This standard deviation is computed from the coordinator activity queuetime histogram and may be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin.
NUM_REQUESTS_ACTIVE	BIGINT	The number of requests that are executing in the service subclass at the time this table function is executed.

Table 51. Information returned for WLM_GET_SERVICE_SUBCLASS_STATS (continued)

Column Name	Data Type	Description
NUM_REQUESTS_TOTAL	BIGINT	The number of requests to finish executing in this service subclass since the last reset. This applies to any request regardless of its membership in an activity. If COLLECT AGGREGATE REQUEST DATA on this service subclass is set to NONE, the value of this column is NULL.
REQUEST_EXEC_TIME_AVG	DOUBLE	Arithmetic mean of the execution times for requests associated with this service subclass since the last reset. Units are milliseconds. If the internally tracked average has overflowed, the value -2 is returned. If COLLECT AGGREGATE REQUEST DATA on this service class is set to NONE, the value of this column is NULL.
REQUEST_EXEC_TIME_STDDEV	DOUBLE	Standard deviation of the execution times for requests associated with this service subclass since the last reset. Units are milliseconds. If COLLECT AGGREGATE REQUEST DATA on this service class is set to NONE, the value of this column is NULL. This standard deviation is computed from the request executetime histogram and may be inaccurate if the histogram has not been properly sized to fit the data. The value of -1 will be returned if any values fall into the last histogram bin.
REQUEST_EXEC_TIME__TOTAL	BIGINT	Sum of the execution times for requests associated with this service subclass since the last reset. Units are milliseconds. If COLLECT AGGREGATE REQUEST DATA on this service class is set to NONE, the value of this column is NULL.

WLM_GET_SERVICE_SUPERCLASS_STATS - Return statistics of service superclasses

This function returns basic statistics of one or more service superclasses.

Syntax

```
►►—WLM_GET_SERVICE_SUPERCLASS_STATS—(—service_superclass_name—,——————————►

►—dbpartitionnum—)——————————————————————————————————————————————►◄
```

The schema is SYSPROC.

Table function parameters

service_superclass_name
: An input argument of type VARCHAR(128) that specifies a valid service superclass name in the same database as the one currently connected to when

calling this function. If the argument is null or an empty string, the data is retrieved for all the superclasses in the database.

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

Authorization

EXECUTE privilege on the WLM_GET_SERVICE_SUPERCLASS_STATS function.

Example

To see all the basic statistics for all the service superclasses on the system, across all database partitions:

```
SELECT SUBSTR(SERVICE_SUPERCLASS_NAME, 1, 26) SERVICE_SUPERCLASS_NAME,
       DBPARTITIONNUM,
       LAST_RESET,
       CONCURRENT_CONNECTION_TOP CONCURRENT_CONN_TOP
FROM TABLE(WLM_GET_SERVICE_SUPERCLASS_STATS('', -2)) as SCSTATS
```

The following is an example of output from this query.

```
SERVICE_SUPERCLASS_NAME    DBPARTITIONNUM ...
-------------------------- -------------- ...
SYSDEFAULTSYSTEMCLASS                   0 ...
SYSDEFAULTMAINTENANCECLASS              0 ...
SYSDEFAULTUSERCLASS                     0 ...
```

Output from this query (continued).

```
... LAST_RESET                 CONCURRENT_CONN_TOP
... -------------------------- -------------------
... 2006-09-05-09.38.44.396788                   0
... 2006-09-05-09.38.44.396795                   0
... 2006-09-05-09.38.44.396796                   1
```

Usage note

The WLM_GET_SERVICE_SUPERCLASS_STATS table function returns one row of data per service superclass and per partition. No aggregation across service superclasses (on a partition), or across partitions (for a service superclass or more) is performed. However, aggregation can be achieved through SQL queries as shown in the example above.

Information returned

Table 52. Information returned for WLM_GET_SERVICE_SUPERCLASS_STATS

Column Name	Data Type	Description
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	Name of the service superclass from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.

Table 52. Information returned for WLM_GET_SERVICE_SUPERCLASS_STATS (continued)

Column Name	Data Type	Description
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that can occur that will trigger a reset of statistics, which will update this timestamp: • The WLM_COLLECT_STATS procedure is called. • The periodic collection and reset process controlled by the WLM_COLLECT_INT configuration parameter causes a collection and reset. • The database is reactivated. • The service superclass for which statistics are being reported was modified and the change was committed. The LAST_RESET timestamp is in local time.
CONCURRENT_CONNECTION_TOP	INTEGER	Highest number of concurrent coordinator connections that has been reached in this class since the last reset.

WLM_GET_WORK_ACTION_SET_STATS - Return work action set statistics

This function returns the statistics for a work action set.

Syntax

```
►►──WLM_GET_WORK_ACTION_SET_STATS──(──work_action_set_name──,──────────────────►

►─dbpartitionnum──)──────────────────────────────────────────────────────────►◄
```

The schema is SYSPROC.

Table function parameters

work_action_set_name
An input argument of type VARCHAR(128) that specifies the specific work action set to return statistics for. If the argument is null or an empty string, statistics are returned for all work action sets.

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

Authorization

EXECUTE privilege on the WLM_GET_WORK_ACTION_SET_STATS function.

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 partition 0, there are 8 activities currently executing (or queued) in the ReadClass, 4 activities currently executing (or queued) in the WriteClass, 2 activities currently executing (or queued) in the LoadClass, and 3 activities currently executing (or queued) that have not been assigned to any work class. Because there is no work action associated with the WriteClass work class, the 4 activities to which it applies are counted in the artificial "*" class along with the 3 activities that were not assigned to any work class.

```
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(WLO_ACT_TOTAL),1,14) AS TOTAL_WLO_ACTS
FROM TABLE(WLM_GET_WORK_ACTION_SET_STATS
     (CAST(NULL AS VARCHAR(128)), -2)) AS WASSTATS
ORDER BY WORK_ACTION_SET_NAME, WORK_CLASS_NAME, PART
```

The following in an example of output from this query.

```
WORK_ACTION_SET_NAME PART WORK_CLASS_NAME LAST_RESET                 TOTAL_WLO_ACTS
-------------------- ---- --------------- -------------------------- --------------
AdminActionSet       0    ReadClass       2005-11-25-18.52.49.343000 8
AdminActionSet       1    ReadClass       2005-11-25-18.52.50.478000 0
AdminActionSet       0    LoadClass       2005-11-25-18.52.49.343000 2
AdminActionSet       1    LoadClass       2005-11-25-18.52.50.478000 0
AdminActionSet       0    *               2005-11-25-18.52.49.343000 7
AdminActionSet       1    *               2005-11-25-18.52.50.478000 0
```

Information returned

Table 53. Information returned for WLM_GET_WORK_ACTION_SET_STATS

Column Name	Data Type	Description
WORK_ACTION_SET_NAME	VARCHAR(128)	The name of the work action set.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that can occur that will trigger a reset of statistics, which will update this timestamp: • The WLM_COLLECT_STATS procedure is called. • The periodic collection and reset process controlled by the WLM_COLLECT_INT configuration parameter causes a collection and reset. • The database is reactivated. • The work action set for which statistics are being reported was modified and the change was committed. The LAST_RESET timestamp is in local time.
WORK_CLASS_NAME	VARCHAR(128)	The name of the work class related to the given work action set. There must be a work action associated with this work class for it to appear in this table. "*" represents an artificial work class created to count all those activities that did not belong to the other work classes for which the user associated one or more work actions.

Table 53. Information returned for WLM_GET_WORK_ACTION_SET_STATS (continued)

Column Name	Data Type	Description
ACT_TOTAL	BIGINT	The number of activities of any nesting level that were assigned to the work class given by WORK_CLASS_NAME.

WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES - Return a list of activities

This function returns the list of all activities that were submitted through the given application on the specified partition and have not yet completed.

Syntax

```
►►──WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES──(──application_handle──,──────────────►

►─dbpartitionnum──)─────────────────────────────────────────────────────────────►◄
```

The schema is SYSPROC.

Table function parameters

application_handle
: An input argument of type BIGINT that specifies an application handle for which a list of activities is returned. If the argument is null, the data is retrieved for all the applications in the database for which the other parameters match.

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

Authorization

EXECUTE privilege on the WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES function.

Example

Once an application handle is identified, it is possible to look up all the activities currently running in this application. For example, suppose an administrator wishes to list the activities of an application whose application handle, determined using the list applications command, was found to be 1:

```
SELECT SUBSTR(CHAR(COORD_PARTITION_NUM),1,5) AS COORD,
     SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
     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(WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES(1, -2)) AS WLOACTS
ORDER BY PART, UOWID, ACTID
```

The following is an example of output from this query.

```
COORD PART UOWID ACTID PARUOWID PARACTID ACTTYPE  NESTING
----- ---- ----- ----- -------- -------- -------- -------
0     0    2     3     -        -        CALL     0
0     0    2     5     2        3        READ_DML 1
```

Information returned

Table 54. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES

Column Name	Data Type	Description
APPLICATION_HANDLE	BIGINT	A system-wide unique ID for the application. On a single-partitioned database, this identifier consists of a 16 bit counter. On a multi-partitioned database, this identifier consists of the coordinating partition number concatenated with a 16 bit counter. In addition, this identifier will be the same on every partition where the application may make a secondary connection.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected.
COORD_PARTITION_NUM	SMALLINT	The coordinator partition of the activity.
LOCAL_START_TIME	TIMESTAMP	The time that this activity began doing work on the partition. It is in local time. This field can be null when an activity has entered the system but is in a queue and has not started executing.
UOW_ID	INTEGER	Unique unit of work identifier within an application. Refers to the original unit of work that the activity started in.
ACTIVITY_ID	INTEGER	Unique activity ID within a unit of work.
PARENT_UOW_ID	INTEGER	Unique unit of work identifier within an application. Refers to the original unit of work that the activity's parent activity started in. Returns null if the activity has no parent activity or at remote partition.
PARENT_ACTIVITY_ID	INTEGER	Unique activity identifier within a unit of work for the parent of the activity whose ID is ACTIVITY_ID. Returns null if the activity has no parent activity or at remote partition.

Table 54. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES (continued)

Column Name	Data Type	Description
ACTIVITY_STATE	VARCHAR(32)	Possible values are: • CANCEL_PENDING - Activity was cancelled when there was no agent actively working on a request for the activity. The next time a request is submitted as part of the activity, the activity will be cancelled and the user who submitted the activity will receive an SQL4725N error. • EXECUTING - Agents are actively working on a request for the activity. • IDLE - There is no agent actively processing a request for the activity. • INITIALIZING - Activity has been submitted, but has not yet started executing. During the initializing state, predictive thresholds are applied to the activity to determine whether or not the activity will be allowed to execute. • QP_CANCEL_PENDING - Same as the CANCEL_PENDING state, but the activity was cancelled by query patroller rather than by the WLM_CANCEL_ACTIVITY procedure. • QP_QUEUED - Activity is queued by Query Patroller. • QUEUED - Activity is queued by a workload management queuing threshold. In a database partitioning feature (DPF) environment, this state might mean that the coordinator agent has made an RPC to the catalog partition to obtain threshold tickets and has not yet received a response. Seeing this state might indicate that the activity has been queued by a workload management queuing threshold or, over short periods of time, can 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 and find out whether this agent's EVENT_OBJECT at the catalog partition has a value of WLM_QUEUE. • TERMINATING - Activity has completed execution and is being removed from the system.

Table 54. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES (continued)

Column Name	**Data Type**	**Description**
ACTIVITY_TYPE	VARCHAR(32)	Possible values are: • CALL • DDL • LOAD • OTHER • READ_DML • WRITE_DML Refer to “Work class work types and SQL statements” in *Workload Manager Guide and Reference* for a description of the different types of SQL statements that are associated with each activity type.
NESTING_LEVEL	INTEGER	This represents the nesting level of this activity. Nesting level is the depth to which this activity is nested within its top-most parent activity.
INVOCATION_ID	INTEGER	This distinguishes one particular invocation of this activity from others at the same nesting level.
ROUTINE_ID	INTEGER	Routine unique identifier.
UTILITY_ID	INTEGER	If the activity is a utility, this is its utility ID. Otherwise, this field is null.
SERVICE_CLASS_ID	INTEGER	Unique identifier of the service class to which this activity belongs.
DATABASE_WORK_ACTION_SET_ID	INTEGER	If this activity has been categorized into a work class of database scope, this column contains the ID of the work class set of which this work class is a member. This column contains null if the activity has not been categorized into a work class of database scope.
DATABASE_WORK_CLASS_ID	INTEGER	If this activity has been categorized into a work class of database scope, this column contains the ID of the work class. This column contains null if the activity has not been categorized into a work class of database scope.
SERVICE_CLASS_WORK_ACTION_SET_ID	INTEGER	If this activity has been categorized into a work class of service class scope, this column contains the ID of the work action set associated with the work class set to which the work class belongs. This column contains null if the activity has not been categorized into a work class of service class scope.

Table 54. Information returned by WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES (continued)

Column Name	Data Type	Description
SERVICE_CLASS_WORK_CLASS_ID	INTEGER	If this activity has been categorized into a work class of service class scope, this column contains the ID of the work class assigned to this activity. This column contains null if the activity has not been categorized into a work class of service class scope.

WLM_GET_WORKLOAD_STATS - Return workload statistics

This function returns workload statistics for every combination of workload name and database partition number.

Syntax

```
►►──WLM_GET_WORKLOAD_STATS──(──workload_name──,──dbpartitionnum──)──────────────►◄
```

The schema is SYSPROC.

Table function parameters

workload_name
An input augment of type VARCHAR(128) that specifies a specific workload for which the statistics are to be returned. If the argument is NULL or an empty string, statistics are returned for all workloads.

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

Authorization

EXECUTE privilege on the WLM_GET_WORKLOAD_STATS function.

Example

An administrator may want to look at the statistics for workloads. She could do so using the following query:

```
SELECT SUBSTR(WORKLOAD_NAME,1,22) AS WL_DEF_NAME,
     SUBSTR(CHAR(DBPARTITIONNUM),1,4) AS PART,
     CONCURRENT_WLO_TOP AS WLO_TOP,
     CONCURRENT_WLO_ACT_TOP AS WLO_ACT_TOP
FROM TABLE(WLM_GET_WORKLOAD_STATS(CAST(NULL AS VARCHAR(128)), -2))
  AS WLSTATS
ORDER BY WL_DEF_NAME, PART
```

The following is an example of output from this query.

```
WL_DEF_NAME            PART WLO_TOP         WLO_ACT_TOP
---------------------- ---- --------------- -------------------
MYUSERWORKLOAD         0                  2                   8
MYUSERWORKLOAD        1                0                0
SYSDEFAULTUSERWORKLOAD 0               1                1
SYSDEFAULTUSERWORKLOAD 1               0                0
```

Here we see that on partition 0, the highest number of concurrent occurrences of the MYUSERWORKLOAD workload was 2 and that the highest number of concurrent activities in either of these workload occurrences was 8.

Usage note

This function returns one row for every combination of workload name and database partition number. No aggregation across workloads or across partitions or across service classes is performed. However, aggregation can be achieved through SQL queries.

Information returned

Table 55. Information returned by WLM_GET_WORKLOAD_STATS

Column Name	Data Type	Description
WORKLOAD_NAME	BIGINT	Name of the workload from which this record was collected.
DBPARTITIONNUM	SMALLINT	Partition number from which this record was collected
LAST_RESET	TIMESTAMP	Time when statistics were last reset. There are four events that can occur that will trigger a reset of statistics, which will update this timestamp: • The WLM_COLLECT_STATS procedure is called. • The periodic collection and reset process controlled by the WLM_COLLECT_INT configuration parameter causes a collection and reset. • The database is reactivated. • The workload for which statistics are being reported was modified and the change was committed. The LAST_RESET timestamp is in local time.
CONCURRENT_WLO_TOP	INTEGER	Highest number of concurrent occurrences of the given workload on this partition since the last reset.
CONCURRENT_WLO_ACT_TOP	INTEGER	Highest number of concurrent activities (including both coordinator and nested) in either executing (which includes idle and waiting) or queued state that has been reached in any occurrence of this workload since last reset. Updated by each workload occurrence at the end of its unit of work.
COORD_ACT_COMPLETED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level assigned to any occurrence of this workload that completed since the last reset. Updated by each workload occurrence at the end of its unit of work.
COORD_ACT_ABORTED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level assigned to any occurrence of this workload that were aborted prior to completion since the last reset. Updated by each workload occurrence at the end of its unit of work.

Table 55. Information returned by WLM_GET_WORKLOAD_STATS (continued)

Column Name	Data Type	Description
COORD_ACT_REJECTED_TOTAL	BIGINT	The total number of coordinator activities at any nesting level assigned to any occurrence of this workload that were rejected prior to execution since the last reset. Updated by each workload occurrence at the end of its unit of work. Activities are counted as rejected when they are prevented from executing by either a prevent execution work action, or a predictive threshold. Note that unlike the column of the same name in the WLM_GET_SERVICE_SUBCLASS_STATS function, this also counts rejections that occur before an activity can be assigned to a service class. An example of such a rejection occurs when an activity violates the ConcurrentWorkloadOccurrences threshold.
WLO_COMPLETED_TOTAL	BIGINT	The number of workload occurrences to complete since last reset.

Chapter 9. Monitor elements

Workload management monitor elements

The following monitor elements provide information about activities, threshold violations, and workload management statistics.

activate_timestamp - Activate timestamp monitor element

The time when an event monitor was activated.

Element identifier
activate_timestamp

Element type
timestamp

Table 56. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activity	event_activity	-
Activity	event_activitystmt	-
Activity	event_activityvals	-
Threshold Violations	event_thresholdviolations	-

Usage

Use this element to correlate information returned by the above 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.

Element identifier
activity_collected

Element type
information

Table 57. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Threshold violations	event_thresholdviolations	-

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. Used with **appl_id** and **uow_id** in an activities event monitor record, this monitor element uniquely identifies an activity that has been collected. Used with **appl_id** and **uow_id** in a threshold violations event monitor record, this monitor element uniquely identifies an activity that has violated a threshold.

Element identifier
activity_id

Element type
information

Table 58. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-
Activities	event_activitystmt	-
Activities	event_activityvals	-
Threshold violations	event_thresholdviolations	-

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 **uow_id** and **agent_id** monitor elements to uniquely identify an activity.

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.

Element identifier
activity_secondary_id

Element type
information

Table 59. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-
Activities	event_activitystmt	-
Activities	event_activityvals	-

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 to which this activity record applies.

Element identifier
activity_type

Element type
information

Table 60. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

Usage

The possible values are:

- LOAD
- READ_DML
- WRITE_DML
- DDL
- CALL
- OTHER

At remote partitions, the value of this monitor element is always OTHER.

act_exec_time - Activity execution time monitor element

Time spent executing at this partition, 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 towards 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 towards 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.

Element identifier
act_exec_time

Element type
time

Table 61. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

Usage

This element can be used alone to know the elapsed time spent executing the activity by DB2 on each partition. This element can also be used together with **time_started** and **time_completed** monitor elements on the coordinator partition 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_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.

Element identifier
act_total

Element type
counter

Table 62. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wcstats	-

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 a actions applied). It can also be used to understand the different types of activities on the system.

arm_correlator - Application response measurement correlator monitor element

Identifier of a transaction in the Application Response Measurement (ARM) standard.

Element identifier
arm_correlator

Element type
information

Table 63. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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.

Element identifier
bin_id

Element type
information

Table 64. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_histogrambin	-

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.

Element identifier
bottom

Element type
information

Table 65. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_histogrambin	-

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.

Element identifier
concurrent_act_top

Element type
watermark

Table 66. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-

Usage

Use this element to know the highest concurrency of activities (including nested activities) reached on a partition 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.

Element identifier
concurrent_connection_top

Element type
watermark

Table 67. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-

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.

Element identifier
concurrent_wlo_act_top

Element type
watermark

Table 68. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wlstats	-

Usage

Use this element to know the highest number of concurrent activities reached on a partition 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.

Element identifier
concurrent_wlo_top

Element type
watermark

Table 69. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wlstats	-

Usage

Use this element to know the highest concurrency of workload occurrences reached on a partition for a workload in the time interval collected.

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.

Element identifier
coord_act_aborted_total

Element type
counter

Table 70. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wlstats	-

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.

Element identifier
coord_act_completed_total

Element type
counter

Table 71. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wlstats	-
Statistics	event_scstats	-

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 partitions.

coord_act_lifetime_top - Coordinator activity lifetime top monitor element

High watermark for coordinator activity lifetime, counted at all nesting levels. Units are 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.

Element identifier
coord_act_lifetime_top

Element type
watermark

Table 72. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wcstats	-
Statistics	event_scstats	-

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_rejected_total - Coordinator activities rejected total monitor element

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.

Element identifier
coord_act_rejected_total

Element type
counter

Table 73. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wlstats	-

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 partition number of the coordinator partition of this activity.

Element identifier
coord_partition_num

Element type
information

Table 74. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-
Threshold violations	event_thresholdviolations	-

Usage

This element allows the coordinator partition to be identified for activities that have records on partitions other than the coordinator.

cost_estimate_top - Cost estimate top monitor element

The 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.

Element identifier
cost_estimate_top

Element type
watermark

Table 75. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

Usage

Use this element to determine the highest DML activity estimated cost reached on a partition for a service class or work class in the time interval collected.

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 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. Units are milliseconds.

Element identifier
coord_act_lifetime_avg

Element type
information

Table 76. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

Usage

Use this statistic to determine the arithmetic mean of the lifetime for coordinator activities associated with a service subclass 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_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. Units are milliseconds. 0 associated with this service subclass that completed or aborted since the last reset. Returns -1 when COLLECT AGGREGATE ACTIVITY DATA of service class is NONE. Units are milliseconds.

Element identifier
coord_act_queue_time_avg

Element type
information

Table 77. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

Usage

Use this statistic to determine the arithmetic mean of the queue time for coordinator activities associated with a service subclass 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_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. Units are milliseconds.

Element identifier
coord_act_exec_time_avg

Element type
information

Table 78. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

Usage

Use this statistic to determine the arithmetic mean of execution time for coordinator activities associated with a service subclass 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.

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.

Element identifier
request_exec_time_avg

Element type
information

Table 79. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-

Usage

Use this statistic to quickly understand the average amount of time that is spent processing each request on a database partition 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.

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 elements returns -1 if no COLLECT AGGREGATE ACTIVITY DATA EXTENDED work action is specified for the work class. Units are milliseconds.

Element identifier
coord_act_est_cost_avg

Element type
information

Table 80. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

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 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_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. Units are milliseconds.

Element identifier
coord_act_interarrival_time_avg

Element type
information

Table 81. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

Usage

Use this statistic to determine the arithmetic mean between arrivals of coordinator activities at nesting level 0 associated with this service subclass 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.

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.

Element identifier
db_work_action_set_id

Element type
information

Table 82. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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.

Element identifier
db_work_class_id

Element type
information

Table 83. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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.

histogram_type - Histogram type monitor element

The type of the histogram, in string format.

There are six histogram types.

CoordActQueueTime
A histogram the time non-nested activities spend queued (for example, in a threshold queue), measured on the coordinator partition.

CoordActExecTime
A histogram of the time non-nested activities spend executing at the coordinator partition. 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.

CoordActLifetime
A histogram of the elapsed lifetime of non-nested activities, measured on the coordinator partition from the time when an activity enters the system until the activity completes execution. Lifetime includes time the activity spends initializing, queued and executing.

CoordActInterArrivalTime
A histogram of the time interval between the arrival of non-nested coordinator activities.

CoordActEstCost
A histogram of the estimated cost of non-nested DML activities.

ReqExecTime
A histogram of request execution times. Includes all requests on both coordinator and non-coordinator partitions including those requests not associated with an activity.

Element identifier
histogram_type

Element type
information

Table 84. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_histogrambin	-

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.

Element identifier
last_wlm_reset

Element type
information

Table 85. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wlstats	-
Statistics	event_wcstats	-
Statistics	event_qstats	-

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_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.

Element identifier
num_threshold_violations

Element type
counter

Table 86. Snapshot Monitoring Information

Snapshot Level	Logical Data Grouping	Monitor Switch
Database	dbase	Basic

For snapshot monitoring, this counter can be reset.

Table 87. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Database	event_db	-

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.

Element identifier
number_in_bin

Element type
information

Table 88. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_histogrambin	-

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 0.

Element identifier
parent_activity_id

Element type
information

Table 89. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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 0.

Element identifier
parent_uow_id

Element type
information

Table 90. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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.

Element identifier
prep_time

Element type
time

Table 91. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

Usage

This element can be used to identify how much of the activity's total lifetime was spent preparing the SQL statement, if this was an SQL activity.

queue_assignments_total - Queue assignments total monitor element

The number of connections or activities that were assigned to this threshold queue since the last reset.

Element identifier
queue_assignments_total

Element type
counter

Table 92. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_qstats	-

Usage

This element can be used to determine the number of activities or connections that were 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.

Element identifier
queue_size_top

Element type
watermark

Table 93. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_qstats	-

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.

Element identifier
queue_time_total

Element type
counter

Table 94. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_qstats	-

Usage

Use this element to gauge the effectiveness of queuing thresholds and to detect when queuing is excessive.

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 database partition for which this information is recorded. On DPF systems, these values may not reflect the correct totals for the whole activity.

Element identifier
rows_fetched

Element type
counter

Table 95. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	Statement

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.

Note: This monitor element reports only the values for the database partition for which this record is recorded. On DPF systems, these values may not reflect the correct totals for the whole activity.

Element identifier
rows_modified

Element type
counter

Table 96. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	Statement

Usage

See the **rows_written** monitor element for details.

rows_returned - Rows returned monitor element

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.

Element identifier
rows_returned

Element type
counter

Table 97. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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 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.

Element identifier
rows_returned_top

Element type
watermark

Table 98. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

Usage

Use this element to know the highest DML activity actual rows returned reached on a partition for a service class 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.

Element identifier
sc_work_action_set_id

Element type
information

Table 99. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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.

Element identifier
sc_work_class_id

Element type
information

Table 100. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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 handle that gives details of an activity's section.

Element identifier
section_env

Element type
information

Table 101. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activitystmt	-

Usage

This element is to be used with future IBM® tools for extracting section information for the activity described in this record

service_class_id - Service class ID monitor element

Unique ID of the service class. Can be used for doing joins with the histogrambins table.

Element identifier
service_class_id

Element type
information

Table 102. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_histogrambin	-
Statistics	event_scstats	-

Usage

Use this element with the **statistics_timestamp** and **partition_number** monitor elements to link histogram bin records with service class statistics records.

service_subclass_name - Service subclass name monitor element

The name of the service subclass to which this activity record or statistics record applies.

Element identifier
service_subclass_name

Element type
information

Table 103. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-
Statistics	event_scstats	-
Statistics	event_qstats	-

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 the service superclass to which this activity record or statistics record applies.

Element identifier
service_superclass_name

Element type
information

Table 104. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-
Statistics	event_scstats	-
Statistics	event_qstats	-

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.

statistics_timestamp - Statistics timestamp monitor element

The time at which this statistics record was generated.

Element identifier
statistics_timestamp

Element type
information

Table 105. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wlstats	-
Statistics	event_wcstats	-
Statistics	event_qstats	-
Statistics	event_histogrambin	-

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.

temp_tablespace_top - Temporary table space top monitor element

The high watermark 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.

Element identifier
temp_tablespace_top

Element type
watermark

Table 106. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_scstats	-
Statistics	event_wcstats	-

Usage

Use this element to determine the highest DML activity system temporary table space usage reached on a partition for a service class 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. If aggregate activity data collection is not enabled for the service class or work class, a value of -1 is returned.

threshold_action - Threshold action monitor element

The action of the threshold to which this threshold violation record applies. Possible values include Stop and Continue.

Element identifier
threshold_action

Element type
information

Table 107. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Threshold violations	event_thresholdviolations	-

Usage

Use this element to determine whether the activity that violated the threshold was stopped when the violation occurred or was allowed to continue executing. If the activity was stopped, the application that submitted the activity will have received an SQL4712N error.

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

Element identifier
threshold_domain

Element type
information

Table 108. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_qstats	-

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.

Element identifier
threshold_maxvalue

Element type
information

Table 109. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Threshold violations	event_thresholdviolations	-

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.

threshold_name - Threshold name monitor element

The unique name of the threshold responsible for this queue.

Element identifier
threshold_name

Element type
information

Table 110. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_qstats	-

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.

Element identifier
threshold_predicate

Element type
information

Table 111. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Threshold violations	event_thresholdviolations	-
Statistics	event_qstats	-

Usage

Use this monitor element in conjunction with other statistics or threshold violation monitor elements for analysis of a threshold violation.

threshold_queuesize - 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.

Element identifier
threshold_queuesize

Element type
information

Table 112. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Threshold violations	event_thresholdviolations	-

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.

Element identifier
thresholdid

Element type
information

Table 113. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Threshold violations	event_thresholdviolations	-
Statistics	event_qstats	-

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.

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 or if the activity was captured while it was in progress.

Element identifier
time_completed

Element type
information

Table 114. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

Usage

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity.

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.

Element identifier
time_created

Element type
information

Table 115. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

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.

Element identifier
time_of_violation

Element type
information

Table 116. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Threshold violations	event_thresholdviolations	-

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.

Element identifier
time_started

Element type
information

Table 117. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

Usage

Use this element in conjunction with other activity history elements for analysis of the behavior of an activity.

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.

Element identifier
top

Element type
information

Table 118. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_histogrambin	-

Usage

Use this element with the corresponding **bottom** element to determine the range of a bin within a histogram.

uow_id - Unit of work ID monitor element

The unit of work ID to which this activity record applies. The unit of work ID is unique within an application handle.

Element identifier
uow_id

Element type
information

Table 119. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-
Activities	event_activitystmt	-
Activities	event_activityvals	-
Threshold violations	event_thresholdviolations	-

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.

wlo_completed_total - Workload occurrences completed total monitor element

The number of workload occurrences to complete since last reset.

Element identifier
wlo_completed_total

Element type
counter

Table 120. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wlstats	-

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.

Element identifier
work_action_set_id

Element type
information

Table 121. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_histogrambin	-
Statistics	event_wcstats	-

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.

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.

Element identifier
work_action_set_name

Element type
information

Table 122. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_qstats	-
Statistics	event_wcstats	-

Usage

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.

Element identifier
work_class_id

Element type
information

Table 123. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wcstats	-
Statistics	event_histogrambin	-

Usage

Use this element in conjunction with other statistics elements for analysis of 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.

Element identifier
work_class_name

Element type
information

Table 124. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_qstats	-
Statistics	event_wcstats	-

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

The ID of the workload to which this activity, application, or workload statistics record belongs.

Element identifier
workload_id

Element type
information

Table 125. Snapshot Monitoring Information

Snapshot Level	Logical Data Grouping	Monitor Switch
Application	appl_info	Basic

Table 126. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wlstats	-
Activities	event_activity	-

Usage

Use this ID to uniquely identify the workload to which this activity, application, or workload statistics record belongs.

workload_name - Workload name monitor element

Name of the workload to which this statistics record applies.

Element identifier
workload_name

Element type
information

Table 127. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Statistics	event_wlstats	-

Usage

Use this element in conjunction with other statistics elements for analysis of a workload.

workload_occurrence_id - Workload occurrence identifier monitor element

The ID of the workload occurrence to which this activity belongs.

Element identifier
: workload_occurrence_id

Element type

Table 128. Event Monitoring Information

Event Type	Logical Data Grouping	Monitor Switch
Activities	event_activity	-

Usage

Use this to identify the workload occurrence that submitted the activity.

Chapter 10. Commands

SET WORKLOAD command

Specifies the workload to which the database connection is to be assigned. This command can be issued prior to 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

```
                        ┌─AUTOMATIC────────────┐
►►──SET WORKLOAD TO─────┴─SYSDEFAULTADMWORKLOAD─┴──────────────────────────────►◄
```

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 *dbadm* or *sysadm* 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 *dbadm* or *sysadm* authority, the connection cannot be assigned to the SYSDEFAULTADMWORKLOAD and an error will be returned. If the SET WORKLOAD TO SYSDEFAULTADMWORKLOAD command is issued prior to connecting to a database, the 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 error will be returned at the beginning of the next unit of work, when the workload reassignment is supposed to take place.

Chapter 11. 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.

Every `x` *wlm_collect_int* 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 event monitor exists, depending on how it was created, the statistics are written either to file or to a table. If it does not exist, the statistics are only reset and not collected.

The collect and reset process is initiated from the catalog partition. The *wlm_collect_int* parameter must be specified on the catalog partition. It is not used on other partitions.

Configuration type
Database

Parameter type
Configurable online

Default [range]
0 [0 (no collection performed), 5 - 32 767]

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's not required, a small interval unnecessarily increases the overhead. Therefore, reduce the interval to capture shorter term behavior, and increase the interval to reduce overhead 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.

Chapter 12. Catalog views

SYSCAT.HISTOGRAMTEMPLATEBINS

Each row represents a histogram template bin.

Table 129. SYSCAT.HISTOGRAMTEMPLATEBINS Catalog View

Column Name	Data Type	Nullable	Description
TEMPLATENAME	VARCHAR (128)	Y	Name of the histogram template.
TEMPLATEID	INTEGER		Identifier for the histogram template.
BINID	INTEGER		Identifier for the histogram template bin.
BINUPPERVALUE	BIGINT		The upper value for a single bin in the histogram template.

SYSCAT.HISTOGRAMTEMPLATES

Each row represents a histogram template.

Table 130. SYSCAT.HISTOGRAMTEMPLATES Catalog View

Column Name	Data Type	Nullable	Description
TEMPLATEID	INTEGER		Identifier for the histogram template.
TEMPLATENAME	VARCHAR (128)		Name of the histogram template.
CREATE_TIME	TIMESTAMP		Time at which the histogram template was created.
ALTER_TIME	TIMESTAMP		Time at which the histogram template was last altered.
NUMBINS	INTEGER		Number of bins in the histogram template, including the last bin that has an unbounded top value.
REMARKS	VARCHAR (254)	Y	User-provided comments, or null.

SYSCAT.HISTOGRAMTEMPLATEUSE

Each row represents a relationship between a workload management object that can use histogram templates and a histogram template.

Table 131. SYSCAT.HISTOGRAMTEMPLATEUSE Catalog View

Column Name	Data Type	Nullable	Description
TEMPLATENAME	VARCHAR (128)	Y	Name of the histogram template.
TEMPLATEID	INTEGER		Identifier for the histogram template.

Table 131. SYSCAT.HISTOGRAMTEMPLATEUSE Catalog View (continued)

Column Name	Data Type	Nullable	Description
HISTOGRAMTYPE	CHAR (1)		The type of information collected by histograms based on this template. • C = Activity estimated cost histogram • E = Activity execution time histogram • I = Activity interarrival time histogram • L = Activity life time histogram • Q = Activity queue time histogram • R = Request execution time histogram
OBJECTTYPE	CHAR (1)		The type of WLM object. • b = Service class • k = Work action
OBJECTID	INTEGER		Identifier of the WLM object.
SERVICECLASSNAME	VARCHAR (128)	Y	Name of the service class.
PARENTSERVICECLASSNAME	VARCHAR (128)	Y	Name of the parent service class.
WORKACTIONNAME	VARCHAR (128)	Y	Name of the work action.
WORKACTIONSETNAME	VARCHAR (128)	Y	Name of the work action set.

SYSCAT.SERVICECLASSES

Each row represents a service class.

Table 132. SYSCAT.SERVICECLASSES Catalog View

Column Name	Data Type	Nullable	Description
SERVICECLASSNAME	VARCHAR (128)		Name of the service class.
PARENTSERVICECLASSNAME	VARCHAR (128)	Y	Service class name of the parent service superclass.
SERVICECLASSID	SMALLINT		Identifier for the service class.
PARENTID	SMALLINT		Identifier for the parent service class for this service class. 0 if this service class is a super service class.
CREATE_TIME	TIMESTAMP		Time when the service class was created.
ALTER_TIME	TIMESTAMP		Time when the service class was last altered.
ENABLED	CHAR (1)		State of the service class. • N = Disabled • Y = Enabled
AGENTPRIORITY	SMALLINT		Thread priority of the agents in the service class relative to the normal priority of DB2 threads. • -20 to 20 (Linux and UNIX) • -6 to 6 (Windows®) • -32768 = not set

Table 132. SYSCAT.SERVICECLASSES Catalog View (continued)

Column Name	Data Type	Nullable	Description
PREFETCHPRIORITY	CHAR (1)		Prefetch priority of the agents in the service class. • H = High • L = Low • M = Medium • Blank = not set
INBOUNDCORRELATOR	VARCHAR (128)	Y	For future use.
OUTBOUNDCORRELATOR	VARCHAR (128)	Y	String used to associate the service class with an operating system workload manager service class.
COLLECTAGGACTDATA	CHAR (1)		Specifies what aggregate activity data should be captured for the service class by the applicable event monitor. • B = Collect base aggregate activity data • E = Collect extended aggregate activity data • N = None
COLLECTAGGREQDATA	CHAR (1)		Specifies what aggregate activity data should be captured for the service class by the applicable event monitor. • B = Collect base aggregate request data • N = None
COLLECTACTDATA	CHAR (1)		Specifies what activity data should be collected by the applicable event monitor. • D = Activity data with details • N = None • V = Activity data with details and values • W = Activity data without details
COLLECTACTPARTITION	CHAR (1)		Specifies where activity data is collected. • C = Database partition of the coordinator of the activity • D = All database partitions
REMARKS	VARCHAR (254)	Y	User-provided comments, or null.

SYSCAT.THRESHOLDS

Each row represents a threshold.

Table 133. SYSCAT.THRESHOLDS Catalog View

Column Name	Data Type	Nullable	Description
THRESHOLDNAME	VARCHAR (128)		Name of the threshold.
THRESHOLDID	INTEGER		Identifier for the threshold.
ORIGIN	CHAR (1)		Origin of the threshold. • U = Threshold was created by a user • W = Threshold was created through a work action set

Table 133. SYSCAT.THRESHOLDS Catalog View (continued)

Column Name	Data Type	Nullable	Description
THRESHOLDCLASS	CHAR (1)		Classification of the threshold. • A = Aggregate threshold • C = Activity threshold
THRESHOLDPREDICATE	VARCHAR (128)		Type of the threshold. Possible values are: • CONCDBC • CONCWCN • CONCWOC • CONNIDLETIME • DBCONN • ESTSQLCOST • ROWSRET • SCCONN • TEMPSPACE • TOTALTIMECONCDBC
THRESHOLDPREDICATEID	SMALLINT		Identifier for the threshold predicate.
DOMAIN	CHAR (2)		Domain of the threshold. • DB = Database • SB = Service subclass • SP = Service superclass • WA = Work action set • WD = Workload definition
DOMAINID	INTEGER		Identifier for the object with which the threshold is associated. This can be a service class, work action or workload unique ID. If this is a database threshold, this value is 0.
ENFORCEMENT	CHAR (1)		Scope of enforcement for the threshold. • D = Database • P = Database partition • W = Workload occurrence
QUEUEING	CHAR (1)		• N = The threshold is not queueing • Y = The threshold is queueing
MAXVALUE	BIGINT		Upper bound specified by the threshold.
QUEUESIZE	INTEGER		If QUEUEING is 'Y', the size of the queue. -1 otherwise.
COLLECTACTDATA	CHAR (1)		Specifies what activity data should be collected by the applicable event monitor. • A = Activity data without details • D = Activity data with details • N = None • V = Activity data with details and values
COLLECTACTPARTITION	CHAR (1)		Specifies where activity data is collected. • C = Database partition of the coordinator of the activity • D = All database partitions

Table 133. SYSCAT.THRESHOLDS Catalog View (continued)

Column Name	Data Type	Nullable	Description
EXECUTION	CHAR (1)		Indicates whether or not execution continues after the threshold has been exceeded. • C = Execution continues • S = Execution stops
ENABLED	CHAR (1)		• N = This threshold is disabled. • Y = This threshold is enabled.
CREATE_TIME	TIMESTAMP		Time at which the threshold was created.
ALTER_TIME	TIMESTAMP		Time at which the threshold was last altered.
REMARKS	VARCHAR (254)	Y	User-provided comments, or null.

SYSCAT.WORKACTIONS

Each row represents a work action that is defined for a work action set.

Table 134. SYSCAT.WORKACTIONS Catalog View

Column Name	Data Type	Nullable	Description
ACTIONNAME	VARCHAR (128)		Name of the work action.
ACTIONID	INTEGER		Identifier for the work action.
ACTIONSETNAME	VARCHAR (128)	Y	Name of the work action set.
ACTIONSETID	INTEGER		Identifier of the work action set to which this work action belongs. This column refers to the ACTIONSETID column in the SYSCAT.WORKACTIONSETS view.
WORKCLASSNAME	VARCHAR (128)	Y	Name of the work class.
WORKCLASSID	INTEGER		Identifier of the work class. This column refers to the WORKCLASSID column in the SYSCAT.WORKCLASSES view.
CREATE_TIME	TIMESTAMP		Time at which the work action was created.
ALTER_TIME	TIMESTAMP		Time at which the work action was last altered.
ENABLED	CHAR (1)		• N = This work action is disabled. • Y = This work action is enabled.

Table 134. SYSCAT.WORKACTIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
ACTIONTYPE	CHAR (1)		The action type that will be performed on each DB2 activity that matches the work class attributes specified in the work class under the matching scope. For this column description, OBJECTTYPE refers to column OBJECTTYPE in SYSCAT.WORKACTIONSETS. • B - Collect basic aggregate activity data. This action type can only be specified if the OBJECTTYPE is 'b' (service class). • C - Allow the execution of any DB2 activity that falls under the work class with which this work action is associated to run and increment the counter for the work class. • D - Collect activity data with details at the database partition of the coordinator of the activity. • E - Collect extended aggregate activity data. This action type can only be specified if the OBJECTTYPE is 'b' (service class). • M - Map to a service subclass. This action type can only be specified if the OBJECTTYPE is 'b' (service class). • P - Prevent the execution of any DB2 activity that falls under the work class with which this work action is associated. • T - The action will be in the form of a threshold. This action type can only be specified if the OBJECTTYPE is 'f' (threshold). • U - Map all activities that have a nesting level of zero and all activities nested under this activity to a service subclass. This action type can only be specified if the OBJECTTYPE is 'b' (service class).
ACTIONTYPE (cont'd)			• V - Collect activity data with details and values at the database partition of the coordinator of the activity. • W - Collect activity data without details at the database partition of the coordinator of the activity. • X - Collect activity data with details at the database partition of the coordinator of the activity and collect activity data at all database partitions. • Y - Collect activity data with details and values at the database partition of the coordinator of the activity and collect activity data at all database partitions. • Z - Collect activity data without details at all database partitions.

Table 134. SYSCAT.WORKACTIONS Catalog View (continued)

Column Name	Data Type	Nullable	Description
REFOBJECTID	INTEGER	Y	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.
REFOBJECTTYPE	VARCHAR (30)		If the ACTIONTYPE is 'M' or 'N', this value is set to 'SERVICE CLASS'; if the ACTIONTYPE is 'T', this value is 'THRESHOLD'; null value otherwise.

SYSCAT.WORKACTIONSETS

Each row represents a work action set.

Table 135. SYSCAT.WORKACTIONSETS Catalog View

Column Name	Data Type	Nullable	Description
ACTIONSETNAME	VARCHAR (128)		Name of the work action set.
ACTIONSETID	INTEGER		Identifier for the work action set.
WORKCLASSSETNAME	VARCHAR (128)	Y	Name of the work class set.
WORKCLASSSETID	INTEGER		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.
CREATE_TIME	TIMESTAMP		Time at which the work action set was created.
ALTER_TIME	TIMESTAMP		Time at which the work action set was last altered.
ENABLED	CHAR (1)		• N = This work action set is disabled. • Y = This work action set is enabled.
OBJECTTYPE	CHAR (1)		• b = Service superclass • Blank = Database
OBJECTNAME	VARCHAR (128)	Y	Name of the service class.
OBJECTID	INTEGER		The identifier of the object to which the work class set (specified by the WORKCLASSSETID) is mapped. If the OBJECTTYPE is blank, the OBJECTID is -1. If the OBJECTTYPE is 'b', the OBJECTID is the ID of the service superclass.
REMARKS	VARCHAR (254)	Y	User-provided comments, or null.

SYSCAT.WORKCLASSES

Each row represents a work class defined for a work class set.

Table 136. SYSCAT.WORKCLASSES Catalog View

Column Name	Data Type	Nullable	Description
WORKCLASSNAME	VARCHAR (128)		Name of the work class.
WORKCLASSSETNAME	VARCHAR (128)	Y	Name of the work class set.
WORKCLASSID	INTEGER		Identifier for the work class.
WORKCLASSSETID	INTEGER		Identifier for the work class set to which this work class belongs. This column refers to the WORKCLASSSETID column in the SYSCAT.WORKCLASSSETS view.
CREATE_TIME	TIMESTAMP		Time at which the work class was created.
ALTER_TIME	TIMESTAMP		Time at which the work class was last altered.
WORKTYPE	SMALLINT		The type of DB2 activity. • 1 = ALL • 2 = READ • 3 = WRITE • 4 = CALL • 5 = DML • 6 = DDL • 7 = LOAD
RANGEUNITS	CHAR (1)		The units to use for the bottom and top range. • C = Cardinality • T = Timerons • Blank = Not applicable
FROMVALUE	DOUBLE	Y	The low value of the range in the units specified by the RANGEUNITS. Null value when RANGEUNITS is blank.
TOVALUE	DOUBLE	Y	The high value of the range in the units specified by the RANGEUNITS. Null value when RANGEUNITS is blank. -1 value is used to indicate no upper bound.
ROUTINESCHEMA	VARCHAR (128)	Y	Schema name of the procedures that are called from the CALL statement. Null value when WORKTYPE is not 4 (CALL) or 1 (ALL).
EVALUATIONORDER	SMALLINT		Uniquely identifies the evaluation order used for choosing a work class within a work class set.

SYSCAT.WORKCLASSSETS

Each row represents a work class set.

Table 137. SYSCAT.WORKCLASSSETS Catalog View

Column Name	Data Type	Nullable	Description
WORKCLASSSETNAME	VARCHAR (128)		Name of the work class set.
WORKCLASSSETID	INTEGER		Identifier for the work class set.

Table 137. SYSCAT.WORKCLASSSETS Catalog View (continued)

Column Name	Data Type	Nullable	Description
CREATE_TIME	TIMESTAMP		Time at which the work class set was created.
ALTER_TIME	TIMESTAMP		Time at which the work class set was last altered.
REMARKS	VARCHAR (254)	Y	User-provided comments, or null.

SYSCAT.WORKLOADAUTH

Each row represents a user, group, or role that has been granted USAGE privilege on a workload.

Table 138. SYSCAT.WORKLOADAUTH Catalog View

Column Name	Data Type	Nullable	Description
WORKLOADID	INTEGER		Identifier for the workload.
WORKLOADNAME	VARCHAR (128)		Name of the workload.
GRANTOR	VARCHAR (128)		Grantor of the privilege.
GRANTORTYPE	CHAR (1)		• U = Grantee is an individual user
GRANTEE	VARCHAR (128)		Holder of the privilege.
GRANTEETYPE	CHAR (1)		• G = Grantee is a group • R = Grantee is a role • U = Grantee is an individual user
USAGEAUTH	CHAR (1)		Indicates whether grantee holds USAGE privilege on the workload. • N = Not held • Y = Held

SYSCAT.WORKLOADCONNATTR

Each row represents a connection attribute in the definition of a workload.

Table 139. SYSCAT.WORKLOADCONNATTR Catalog View

Column Name	Data Type	Nullable	Description
WORKLOADID	INTEGER		Identifier for the workload.
WORKLOADNAME	VARCHAR (128)		Name of the workload.
CONNATTRTYPE	VARCHAR (30)		Type of the connection attribute. • 1 = APPLNAME • 2 = SYSTEM_USER • 3 = SESSION_USER • 4 = SESSION_USER GROUP • 5 = SESSION_USER ROLE • 6 = CURRENT CLIENT_USERID • 7 = CURRENT CLIENT_APPLNAME • 8 = CURRENT CLIENT_WRKSTNNAME • 9 = CURRENT CLIENT_ACCTNG

Table 139. SYSCAT.WORKLOADCONNATTR Catalog View (continued)

Column Name	Data Type	Nullable	Description
CONNATTRVALUE	VARCHAR (1000)		Value of the connection attribute.

SYSCAT.WORKLOADS

Each row represents a workload.

Table 140. SYSCAT.WORKLOADS Catalog View

Column Name	Data Type	Nullable	Description
WORKLOADID	INTEGER		Identifier for the workload.
WORKLOADNAME	VARCHAR (128)		Name of the workload.
EVALUATIONORDER	SMALLINT		Evaluation order used for choosing a workload.
CREATE_TIME	TIMESTAMP		Time at which the workload was created.
ALTER_TIME	TIMESTAMP		Time at which the workload was last altered.
ENABLED	CHAR (1)		• N = This workload is disabled. • Y = This workload is enabled.
ALLOWACCESS	CHAR (1)		• N = A UOW associated with this workload will be rejected. • Y = A unit of work (UOW) associated with this workload can access the database.
SERVICECLASSNAME	VARCHAR (128)		Name of the service subclass to which a unit of work (associated with this workload) is assigned.
PARENTSERVICECLASSNAME	VARCHAR (128)	Y	Name of the service superclass to which a unit of work (associated with this workload) is assigned.
COLLECTAGGACTDATA	CHAR (1)		Specifies what aggregate activity data should be captured for the workload by the applicable event monitor. • N = None
COLLECTACTDATA	CHAR (1)		Specifies what activity data should be collected by the applicable event monitor. • D = Activity data with details • N = None • V = Activity data with details and valuesApplies when the COLLECT column is set to 'C' • W = Activity data without details
COLLECTACTPARTITION	CHAR (1)		Specifies where activity data is collected. • C = Database partition of the coordinator of the activity • D = All database partitions
EXTERNALNAME	VARCHAR (128)	Y	Reserved for future use.
REMARKS	VARCHAR (254)	Y	User-provided comments, or null.

Part 6. Appendixes

Appendix A. Workload management DDL statement considerations

DB2 workload management DDL statements consist of the CREATE, ALTER, and DROP statements you use to work with service classes, workloads, work class sets, work action sets, thresholds, and histograms.

The workload management DDL statements are as follows:
- CREATE WORKLOAD, ALTER WORKLOAD, and DROP WORKLOAD
- GRANT USAGE ON WORKLOAD and REVOKE USAGE ON WORKLOAD
- CREATE SERVICE CLASS, ALTER SERVICE CLASS, and DROP SERVICE CLASS
- 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 THRESHOLD, ALTER THRESHOLD, and DROP THRESHOLD
- 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 allowed at a time across all database partitions. 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.

Many database objects have owners, and these owners have authority to alter the objects that they own. Unlike most objects, workload management objects do not have owners, because this could cause unpredictable problems. For example, if a resource allocation setting is changed for a service class, the change affects not only the service class itself but also other service classes of the same tier. For example, assume that a service superclass has two user-defined service subclasses: A and B, and that each service subclass has a different owner. Initially, the prefetch priority setting is medium for the default service subclass and for the two service subclasses A and B. If the owner of service subclass A were to change its prefetch priority to high and many prefetch requests come from this service subclass, connections to service subclass B and the default subclass would be *starved* for prefetcher services and the performance of activities running in these service subclasses might suffer. For reasons such as this, DB2 workload management objects do not have owners.

Appendix B. Integration of DB2 workload management and the AIX Workload Manager

When DB2 workload management is used in conjunction with the AIX Workload Manager, additional capabilities are available. The AIX Workload Manager can be used to control the amount of CPU allocated to each service class.

Options include setting a minimum, maximum, or relative proportion share of CPU for each service class. Mapping between DB2 service classes and AIX Workload Manager service classes is specified in the definition of the DB2 service class using the OUTBOUND CORRELATOR option of the CREATE SERVICE CLASS or the ALTER SERVICE CLASS statement. DB2 service classes are the sole point of integration between DB2 workload management and the AIX Workload Manager.

This topic describes:

- Recommended mappings between DB2 service classes and AIX Workload Manager service classes
- Instructions for defining the mappings between DB2 service classes and AIX Workload Manager service classes
- Setting CPU controls on the AIX Workload Manager service classes

Recommended mappings between DB2 service classes and AIX Workload Manager service classes

You can map DB2 service classes to the AIX Workload Manager service classes in any way you want. However, you should try to start with a 1:1 mapping of DB2 service classes to the AIX Workload Manager service classes. A 1:1 mapping is a good starting point because activities in a DB2 service class should have more or less the same performance goal. The most direct way to meet that goal is to adjust the AIX resources allocated to that AIX Workload Manager service class. By having a 1:1 mapping between a DB2 service class and an AIX Workload Manager service class, you can adjust the AIX resources for each service class individually.

The following figure shows the integration of the DB2 workload manager 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.

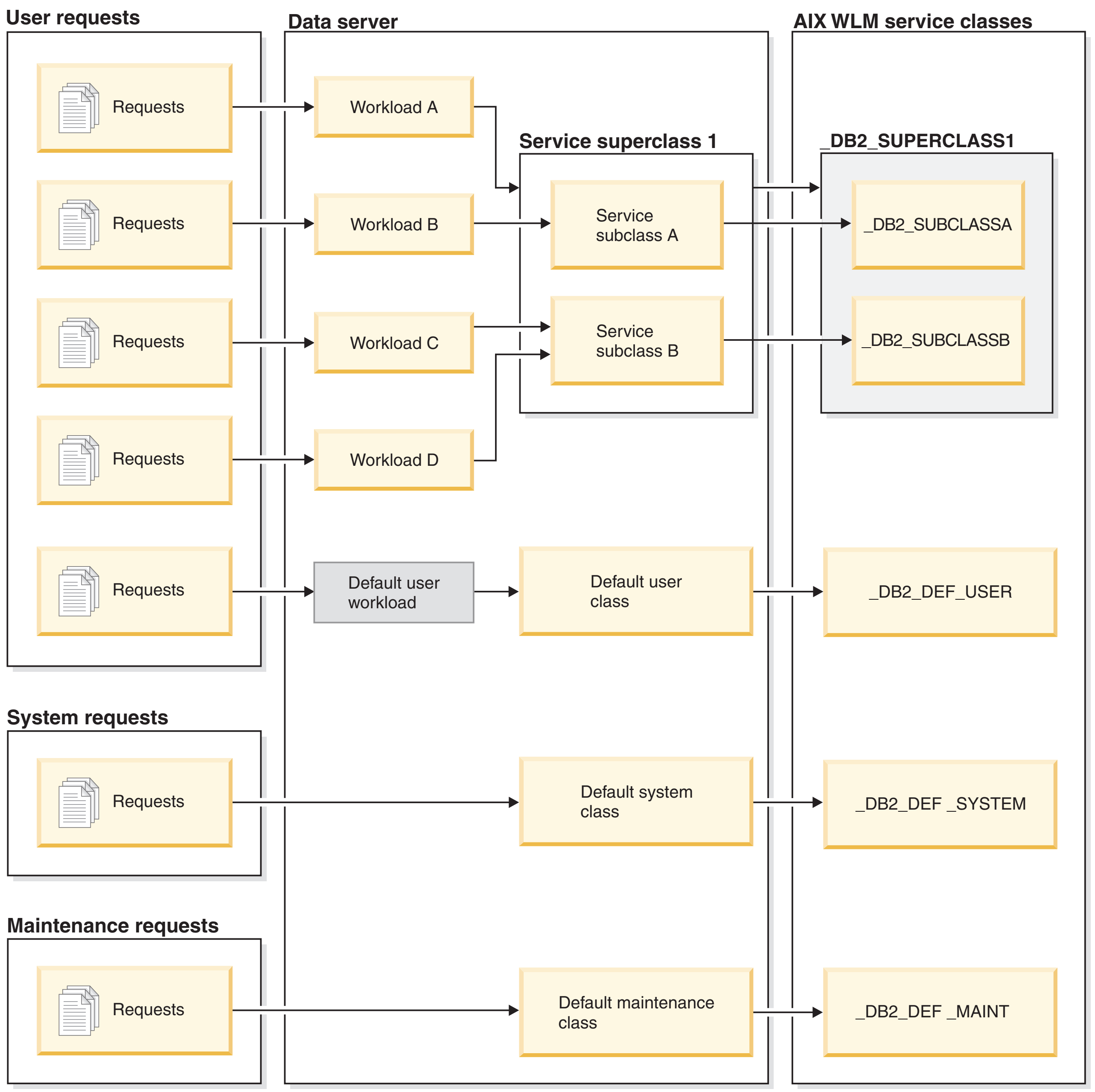


Figure 28. Integration of the DB2 workload manager with the AIX Workload Manager

When a DB2 environment consists of a single database in a single DB2 instance, like the example portrayed in the previous figure, it is possible to map directly between DB2 service classes and AIX Workload Manager service 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.

The following figure shows that, in situations where the DB2 environment consists of multiple databases and DB2 instances, four levels are candidates for resource control.

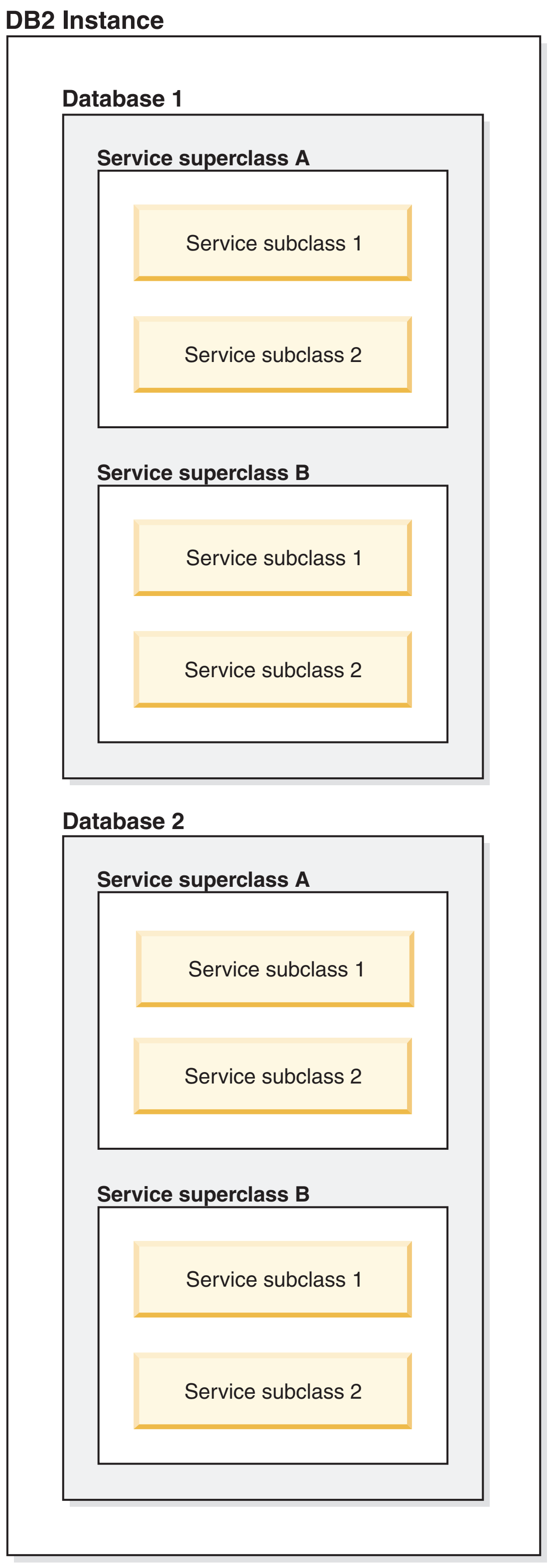


Figure 29. Resource control levels in a DB2 environment

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 service classes at any time. Some sample configurations follow.

The following figure shows one way to achieve the 1:1 mapping in the case with multiple databases, each with superclasses. Here, each database has its own AIX Workload Manager superclass and each DB2 service superclass is mapped to an AIX Workload Manager subclass.

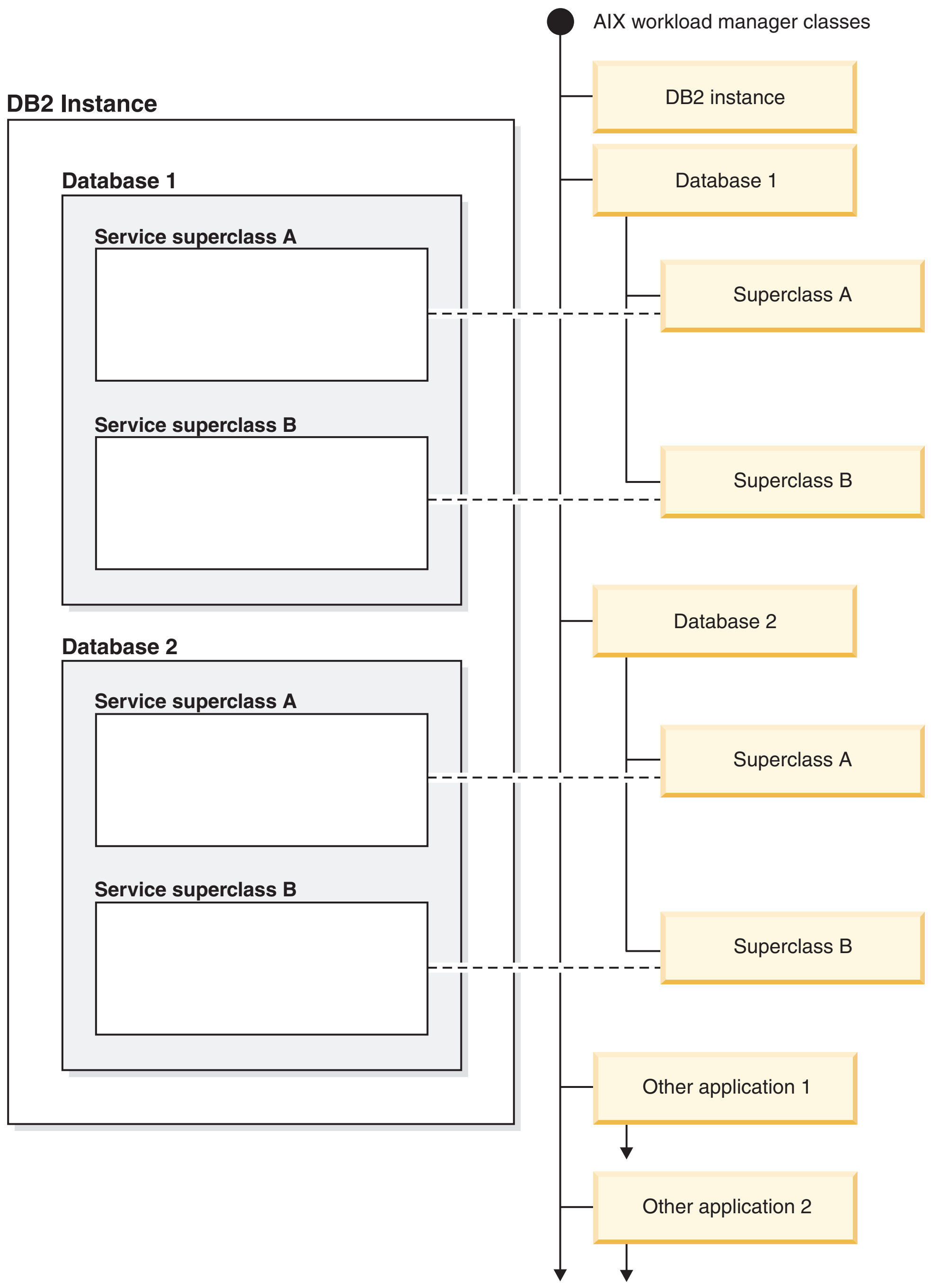


Figure 30. DB2 service classes mapped to AIX service classes (with DB2 service superclasses only)

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 is mapped 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.

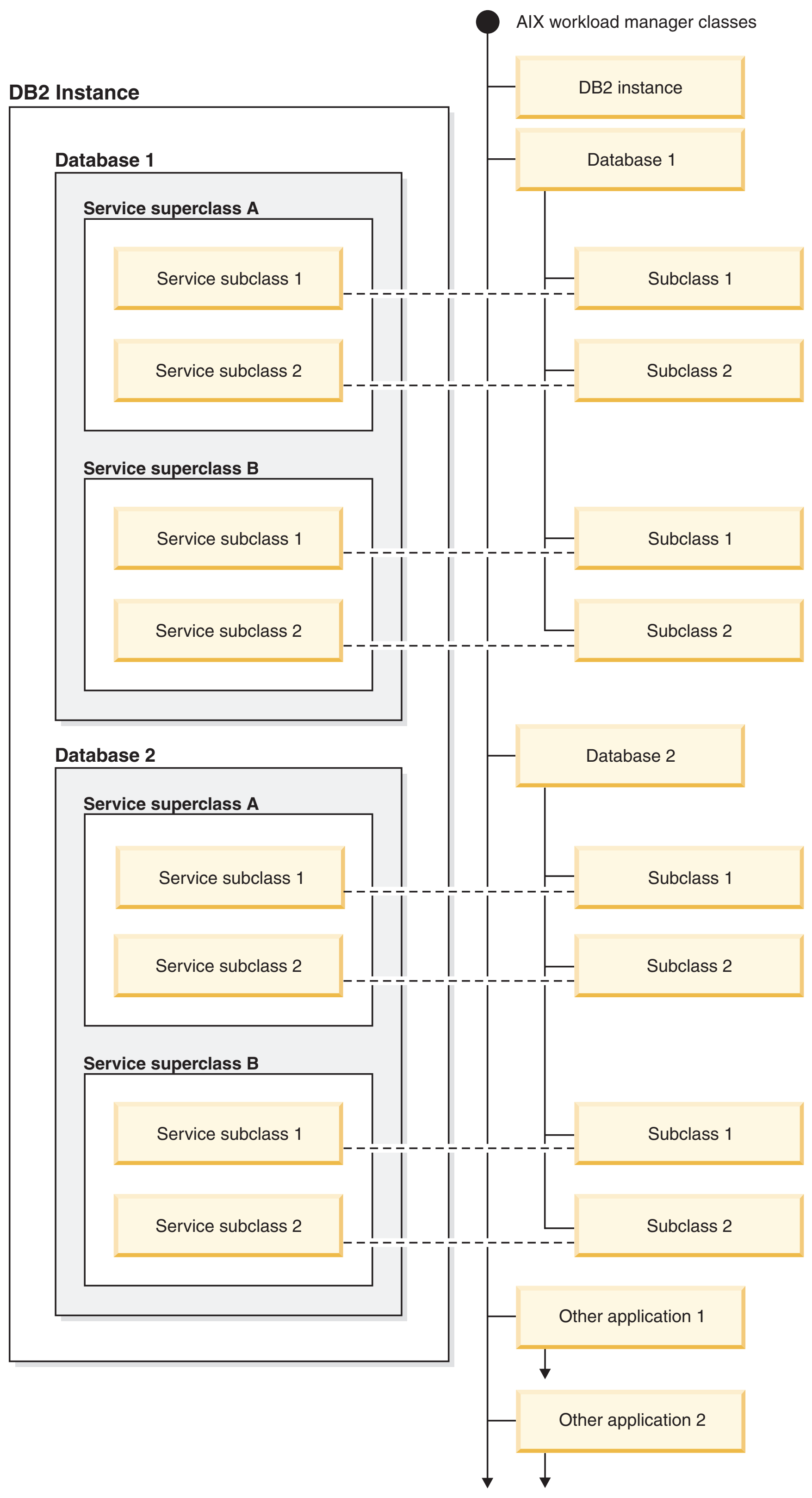


Figure 31. DB2 service classes mapped to AIX Workload Manager classes (with DB2 service subclasses)

Defining mappings between DB2 service classes and AIX Workload Manager service classes

Mapping between DB2 service classes and AIX Workload Manager service 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 service 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 service 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.

The following points explain how the AIX Workload Manager handles work from the DB2 data server given the change in DB2 Version 9.5 from a process model to a threaded model on UNIX and Linux. 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 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, the DB2 workload manager cannot change the AIX workload management class of a thread using tagging. This restriction makes any integration of the DB2 workload manager 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.
- AIX 5.3.H or later is required.

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 CPU controls on the AIX Workload Manager service classes

The AIX Workload Manager can be used to control the amount of CPU allocated to each service class. Options include setting a minimum, maximum, or relative proportion share of CPU for each service class.

When integrating the AIX Workload Manager with DB2 workload management, only CPU allocation control is supported. You should not set memory and I/O settings for the AIX service 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 new 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.

If you use AIX to control the amount of CPU allocated to a service class, do not also change the agent priority setting for that DB2 service class. Use only one of these mechanisms to govern the access to CPU resources. You cannot set both the AGENT PRIORITY and the OUTBOUND CORRELATOR value for a service class. See "CPU priority and DB2 service classes" on page 30 for more information.

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 partitions. This situation can lead to poor overall throughput for connections in an AIX service class.

Appendix C. Processing of stored procedures in a workload management solution

Each invocation of a stored procedure is processed as one activity. All database activities issued by the stored procedure are processed as child activities of the stored procedure. If a mapping work action is applied to a stored procedure, depending on its configuration, child activities of a stored procedure can run in the same service subclass or different service subclasses than the parent activity.

Activity concurrency thresholds are applied to the stored procedure itself and its child activities. If the execution of the stored procedure is queued, none of its child activities can proceed. However, when a stored procedure starts running, child activities might be queued.

For stored procedures that run in fenced mode, you can see the process ID and thread ID of the processes or threads of the stored procedure using the WLM_GET_SERVICE_CLASS_AGENTS table function.

Appendix D. Naming rules

Rules exist for the naming of all objects, users and groups. Some of these rules are specific to the platform you are working on.

For example, there is a rule regarding the use of upper and lowercase letters in a name.
- On UNIX platforms, names must be in lowercase.
- On Windows platforms, names can be in upper, lower, and mixed-case.

Unless otherwise specified, all names can include the following characters:
- A through Z. When used in most names, characters A through Z are converted from lowercase to uppercase.
- 0 through 9.
- ! % ( ) { } . – ^ ~ _ (underscore) @, #, $, and space.
- \ (backslash).

Names cannot begin with a number or with the underscore character.

Do not use SQL reserved words to name tables, views, columns, indexes, or authorization IDs.

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 need to follow the rules forced on specific operation systems by the related systems. For example, on Linux and UNIX platforms, allowed characters for user names and primary 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.

You also need to consider object naming rules, naming rules in an NLS environment, and naming rules in a Unicode environment.

Restrictions on the AUTHID identifier: Version 9.5, and later, of the DB2 database system allows you to have an 128-byte authorization ID, but when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply (for example, a limitation to 8 or 30 character user IDs and 30 character group names). Therefore, while you can grant an 128-byte authorization ID, it is not possible to connect as a user that has that authorization ID. If you write your own security plugin, you should be able to take full advantage of the extended sizes for the authorization ID. For example, you can give your security plugin a 30-byte user ID and it can return an 128-byte authorization ID during authentication that you are able to connect with.

Appendix E. 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, with the exception of security administrator (SECADM) authority. For example, a role can be granted any of the following authorities and privileges:

- DBADM, 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.
- A role cannot be granted security administrator (SECADM) authority.
- 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 F. 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 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. In other words, 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 problem, 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 G. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com®.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. English DB2 Version 9.5 manuals in PDF format and translated versions can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The DB2 Information Center is updated more frequently than either the PDF or the hard-copy books.

Table 141. DB2 technical information

Name	Form Number	Available in print
Administrative API Reference	SC23-5842-01	Yes
Administrative Routines and Views	SC23-5843-01	No
Call Level Interface Guide and Reference, Volume 1	SC23-5844-01	Yes
Call Level Interface Guide and Reference, Volume 2	SC23-5845-01	Yes
Command Reference	SC23-5846-01	Yes
Data Movement Utilities Guide and Reference	SC23-5847-01	Yes
Data Recovery and High Availability Guide and Reference	SC23-5848-01	Yes
Data Servers, Databases, and Database Objects Guide	SC23-5849-01	Yes
Database Security Guide	SC23-5850-01	Yes
Developing ADO.NET and OLE DB Applications	SC23-5851-01	Yes
Developing Embedded SQL Applications	SC23-5852-01	Yes
Developing Java Applications	SC23-5853-01	Yes
Developing Perl and PHP Applications	SC23-5854-01	No
Developing User-defined Routines (SQL and External)	SC23-5855-01	Yes
Getting Started with Database Application Development	GC23-5856-01	Yes
Getting Started with DB2 installation and administration on Linux and Windows	GC23-5857-01	Yes
Internationalization Guide	SC23-5858-01	Yes
Message Reference, Volume 1	GI11-7855-00	No
Message Reference, Volume 2	GI11-7856-00	No
Migration Guide	GC23-5859-01	Yes
Net Search Extender Administration and User's Guide	SC23-8509-01	Yes
Partitioning and Clustering Guide	SC23-5860-01	Yes
Query Patroller Administration and User's Guide	SC23-8507-00	Yes
Quick Beginnings for IBM Data Server Clients	GC23-5863-01	No

Table 141. DB2 technical information (continued)

Name	Form Number	Available in print
Quick Beginnings for DB2 Servers	GC23-5864-01	Yes
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC23-8508-01	Yes
SQL Reference, Volume 1	SC23-5861-01	Yes
SQL Reference, Volume 2	SC23-5862-01	Yes
System Monitor Guide and Reference	SC23-5865-01	Yes
Troubleshooting Guide	GI11-7857-01	No
Tuning Database Performance	SC23-5867-01	Yes
Visual Explain Tutorial	SC23-5868-00	No
What's New	SC23-5869-01	Yes
Workload Manager Guide and Reference	SC23-5870-01	Yes
pureXML Guide	SC23-5871-01	Yes
XQuery Reference	SC23-5872-01	No

Table 142. DB2 Connect-specific technical information

Name	Form Number	Available in print
Quick Beginnings for DB2 Connect Personal Edition	GC23-5839-01	Yes
Quick Beginnings for DB2 Connect Servers	GC23-5840-01	Yes
DB2 Connect User's Guide	SC23-5841-01	Yes

Table 143. Information Integration technical information

Name	Form Number	Available in print
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	Yes
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	Yes
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-01	No
Information Integration: SQL Replication Guide and Reference	SC19-1030-01	Yes
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	Yes

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* DVD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation DVD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation DVD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  1. Locate the contact information for your local representative from one of the following Web sites:
     - The IBM directory of world wide contacts at www.ibm.com/planetwide
     - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the ″About this site″ link.
  2. When you call, specify that you want to order a DB2 publication.
  3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 287.

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

To invoke SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

For DB2 Version 9.5 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

- To display topics in your preferred language in the Internet Explorer browser:
  1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     – To add a new language to the list, click the **Add...** button.

       Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
     – To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.
- To display topics in your preferred language in a Firefox or Mozilla browser:
  1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     – To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
     – To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. Non-Administrative and Non-Root DB2 Information Centers always run in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you would like to install, you can use the Update feature to obtain and install them

   Note: If your environment requires installing the DB2 Information Center updates on a machine that is not connected to the internet, you have to mirror the update site to a local file system using a machine that is connected to the internet and has the DB2 Information Center installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.
   If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.
3. Stop the stand-alone Information Center, and restart the DB2 Information Center on your computer.

Note: On Windows Vista, the commands listed below must be run as an administrator. To launch a command prompt or graphical tool with full administrator privileges, right-click on the shortcut and then select **Run as administrator**.

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv95 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the <Program Files>\IBM\DB2 Information Center\Version 9.5 directory, where `<Program Files>` represents the location of the Program Files directory.
     c. Navigate from the installation directory to the doc\bin directory.
     d. Run the help_start.bat file:
        ```
        help_start.bat
        ```
   - On Linux:

   a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the /opt/ibm/db2ic/V9.5 directory.
   b. Navigate from the installation directory to the doc/bin directory.
   c. Run the help_start script:

      ```
      help_start
      ```

   The systems default Web browser launches to display the stand-alone Information Center.
3. Click the **Update** button (🔄). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
   - On Windows, navigate to the installation directory's doc\bin directory, and run the help_end.bat file:

     ```
     help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use `Ctrl-C` or any other method to terminate help_start.bat.
   - On Linux, navigate to the installation directory's doc/bin directory, and run the help_end script:

     ```
     help_end
     ```

     Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the help_start script.
7. Restart the DB2 Information Center.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv95 start
     ```

The updated DB2 Information Center displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click on the title.

"pureXML™" in *pureXML Guide*
: Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

"Visual Explain" in *Visual Explain Tutorial*
: Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation
: Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site
: Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

  Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/support.html

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

Appendix H. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give 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 (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This document may provide links or references to non-IBM Web sites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources that may be referenced, accessible from, or linked from this document. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or

Trademarks

Index

Printed in USA

SC23-5870-01

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

Workload Manager Guide and Reference