IBM® DB2® for Linux®, UNIX®, and Windows®

Best Practices

Implementing DB2® Workload Management in a Data Warehouse

Authors

Paul Bird

Senior Technical Staff Member

Optim & DB2® Development

IBM Toronto Laboratory

Rimas P. Kalesnykas

DB2 Information Developer

IBM Toronto Laboratory

Contents

Executive summary

The objective of this document is to guide anyone new to the implementation of workload management within the DB2 for Linux, UNIX, and Windows product through the step-by-step process needed to establish an initial configuration that is designed to help ensure the stability and predictability of the database system as a whole.

The following are the two main lessons learned from experience with regards to workload management:

1. Having a workload management plan or configuration in place for your system provides the following significant advantages:
   - A database system cannot become swamped and unresponsive due to low priority, complex work consuming too much resource
   - The root cause of many issues in DB2 environments are resolved with the implementation of workload management principles
2. A workload management configuration that is too complex makes a system difficult to monitor and manage
   - The "Keep It Simple, Silly" (KISS) principle is still the best approach, and complexity should be added only when needed

This document presents a set of definitions representing the different stages of maturity for a workload management configuration in a DB2 for Linux, UNIX, and Windows database. These stages range from stage 0 through to the advanced stage 3 configuration. A specific configuration template and process is provided as part of these best practices to enable customers to progress from a stage 0 configuration to a stage 2 configuration. General descriptions and advice are also given about common stage 3 scenarios.

It is recommended that all DB2 customers, with a data warehouse, implement at least a stage 2 workload management configuration and this goal is the primary focus of this document. A stage 2 configuration focuses on stabilizing the overall system behavior. Some customers might require a more advanced, stage 3 configuration to address their specific application performance objectives or to support their business and IT philosophies regarding how users are serviced.

Introduction

The focus of this document is to describe how best to implement a successful workload management solution using DB2 for Linux, UNIX, and Windows, Version 9.7.4 or higher. The contents of this document reflect the latest experiences of IBM® field personnel and customers within the data warehouse arena from which the vast majority of reported feedback for DB2 workload management has been received.

Using a staged approach, this document guides you through the steps needed to implement the best practices workload management configuration on DB2 for Linux, UNIX, and Windows with sufficient controls to help ensure a stable, predictable system for most data warehouse environments. This initial configuration is intended to be a good base upon which additional tuning and configuration changes can be implemented, as needed, for you to achieve your specific workload management objectives.

The document assumes a novice beginner and describes the individual steps and mechanisms at each point. A more experienced user can condense many of the listed steps to move from stage 1 to stage 2, making the transition in days of elapsed time rather than weeks as the suggested timeline indicates in a later section.

Although you can use SQL DDL statements to directly interact with the DB2 database manager, the IBM Optim™ Performance Manager 4.1.1 product provides a simplified interface through its detailed configuration editor which can be used to make the interactions less onerous.

The steps outlined in this document are focused on the efficiency of the system as a whole, regardless of where the work itself comes from. It is important to note that achieving the goal of a stable system might not necessarily also result in the achievement of any individual application service-level agreement (SLA) or specific performance objectives for queries. These more granular objectives might require subsequent changes to the workload management configuration, such as outlined in the section on stage 3 scenarios, which is outside the main scope of this document.

This paper is not a tutorial on DB2 workload management capabilities and does not attempt to provide comprehensive guidance in addressing all possible scenarios where DB2 workload management might be employed. It also does not cover all features within the DB2 product that might be of use in controlling resource consumption. The scope of this paper is focused on describing the system stabilization approach in some detail and provides some general guidance for common advanced scenarios.

A short review of DB2 workload management

A general definition for database workload management is the process or act of monitoring and controlling work executing on a database system in order to make efficient use of system resources in addition to achieving any performance objectives assigned to that work. Such a broad definition also encompasses much of the original effort put into designing and implementing a successful database system in the first place because they share the same end goals.

A more pragmatic definition for workload management might be the process or act of monitoring and controlling the competition between work for system resources by imposing the business priorities and performance objectives, identified for each of the competitors, onto the decisions made by the database system. It is important that workload management is viewed as a complement, not a substitute, for following the other best practices provided for designing and implementing a successful database. You cannot use workload management techniques to put a good-looking façade over a bad design.

Using this definition, one could also assert that any effort to implement workload management needs to encompass both monitoring and control aspects because both are critical to a successful implementation. Monitoring provides you with the operational awareness of how the database and individual workloads are progressing so that you can keep things running smoothly. Control gives you the tools you need to manage resource consumption by the different workloads running on the database.

Workload management is an item of concern for almost all databases, not just those databases classified as data warehouse systems. Workload conflicts and resource contentions do not discriminate based on the (sometimes arbitrary) category assigned to your particular database. Any time there is more than one type of workload or class of users accessing a database, you have the potential for conflict or a desire to provide differentiation in the level of service that each receives from the database. Unless your database is host to only a single workload with a single class of user, then some form of workload management is of interest to you, even if it is just the monitoring aspect.

DB2 workload management and DB2 workload manager

From a DB2 for Linux, UNIX, and Windows perspective, DB2 Version 9.5 contained a major investment in workload management. A new infrastructure and a set of capabilities was provided to better enable the implementation of successful workload management solutions in a wide variety of customer environments. This infrastructure is generically referred to as the DB2 workload management infrastructure. It incorporates both a core set of capabilities and a set of capabilities that are only available under license; the licensed set of capabilities are referred to as DB2 workload manager (DB2 WLM).

The core technology is always present and active when a DB2 database is active such that all work executing within a DB2 database is executing within a specific workload management configuration, even if it is only the default configuration provided by the DB2 database manager. When you install DB2 Version 9.5 or later, you are automatically using the new DB2 workload management capabilities in the form of the default workload and service classes that are installed as part of every DB2 database; the default workload management configuration is available to all DB2 database manager customers. You are able to use all of the workload management monitoring capabilities with this default configuration.

The set of DB2 workload manager capabilities, controlled by license, allow you to customize the default DB2 workload management configuration to better reflect the organization and priorities of your business. You need to have the license prerequisite for DB2 workload manager in DB2 Version 9.5 and Version 9.7 to create any or all of the following:

- A DB2 workload
- A DB2 service class

- A DB2 threshold
- A DB2 work action set

For detailed information about the significant enhancements to DB2 workload management and the monitoring capabilities associated with workloads, thresholds, and service classes for DB2 for Linux, UNIX, and Windows, Version 9.7, see Appendix A, "DB2 workload management and monitoring highlights for DB2 for Linux, UNIX, and Windows Version 9.7," on page 71.

Prerequisite concepts and terminology

The reader of this document, at a minimum, must have a conceptual familiarity with the key components that comprise workload management function in the DB2 product and the terminology used throughout this best practices paper.

It is not in the scope of this paper to give detailed background information about workload management with DB2 workload manager. This detailed information can be obtained from the DB2 Version 9.7 Information Center.

The following are the key components of DB2 workload management functions:

activity
A database entity that uses database resources during its lifetime, which can span one or more requests. A cursor and a procedure are examples of activities. The DB2 workload management infrastructure explicitly recognizes a specific subset from the domain of all possible database activities, specifically the LOAD utility, the CALL SQL statement, all DML SQL statements, and all DDL SQL statements. For these recognized activities, full support for monitoring and controlling them is provided by DB2 workload management. For all other *unrecognized* activities, they are mapped directly to the service class defined in the workload definition and they are not acted upon by any of the other capabilities within DB2 workload management.

DB2 workload
One or more database activities or requests working within a service class or a database.

A database transaction is mapped to a DB2 workload based on a user-defined set of transaction attributes (connection attributes are considered a part of database transaction attributes). The workload acts as the primary point of monitoring and control for transactions submitting work to a database. Also, the workload directs any incoming work from mapped transactions to a specific DB2 service class for execution. The terms *transaction* and *unit of work* are equivalent.

DB2 service class
An entity that acts as the primary point for monitoring, control, and resource assignment for work executing within a database.

There are two levels of service classes that together form a simple, hierarchical model:

Superclass
A grouping mechanism for subclasses within a database. Resources and settings of a service superclass are shared by all related service subclasses.

Subclass
A grouping mechanism for database activities within a service superclass. All user work processed in a database executes within the context of a DB2 service subclass.

DB2 threshold
A threshold is a mechanism provided as the primary method for automatic monitoring and enforcement of acceptable behavior for the execution of work. Some thresholds monitor specific aspects of work execution such as resource consumption or time spent executing, and other thresholds control the degree of concurrency allowed at any one time. Thresholds can be applied at different levels of a database including per workload and per service class.

DB2 work action set
The work action set, with its companion the work class set, provides the ability to discriminate between different types of database activities and treat them differently for monitoring purposes, or control purposes, or both. Thresholds can be applied using a work action set on a workload or a database; when a work action set is defined on a service superclass, incoming work can be mapped to service subclasses with finer granularity.

Identification of best practices configuration steps

Wherever specific best practices configuration steps are provided in this paper, the steps are marked with a light bulb icon in the left margin, as shown here, and each step has been classified into the following types for your convenience:

Data collection step:
Create event monitors and collect monitoring data that is required for analysis. This step includes changes to workload management DDL related to specifying what monitoring data to collect.

Analysis step:
View and analyze the collected monitoring data. Make workload management tuning decisions.

Implementation step:
Alter a workload management configuration to change how work is controlled or classified, guided by your analysis of the collected monitoring data.

Rationale: DB2 workload management best practices goals and objectives

Overview

At some point in time, most data warehouses run into a situation where the demands or performance expectations of the business exceed the capacity of the database system. Not all organizations want to face this fact or make the business priority decisions needed to survive it. These decisions involve either investing in a larger, more powerful system, or deciding how much of the existing system resources are to be consumed by each of the different types of work running on that system. Sometimes, the business just needs to survive on the existing, constrained system in order to buy time for the new system to be funded and installed.

It is at this point where workload management enters the picture. With advanced planning, workload management capabilities make it possible to dictate the behavior of the system and make it more predictable, even when demand exceeds capacity.

There are two perspectives that need to be taken into account when planning and designing your workload management system, as depicted in Figure 1 on page 8. The business perspective encompasses processes and applications being used by different parts of your business with different demands and performance expectations or objectives. The system perspective deals with the realities of managing your system efficiently.

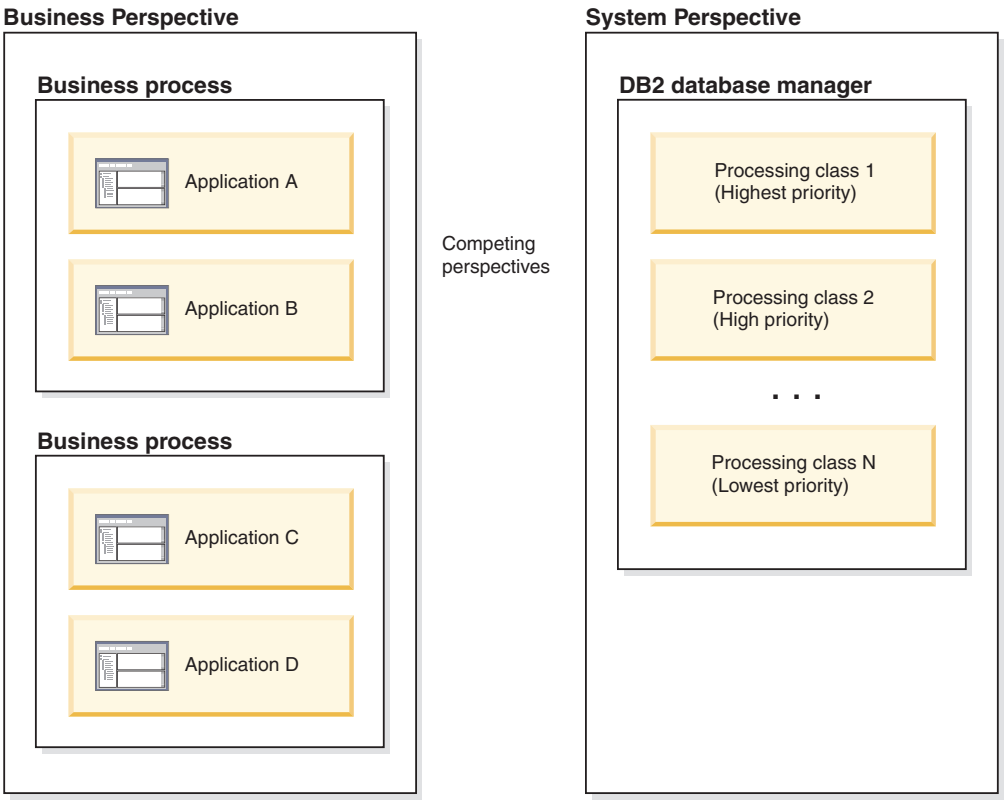


Figure 1. Business versus system perspectives

The challenge of workload management is to map the business perspective to the system perspective while, at the same time, meeting the business performance objectives. The best strategy is to first focus on managing the system capacity as efficiently as possible, regardless of where the work comes from. After you have a stable, predictable system configuration, then you can begin to look at controlling access to the system based on business priorities.

Achieving a stable, predictable system

The primary goal and main theme of this workload management best practices paper is to document the required configuration to achieve a stable, predictable system. From this foundation, you can significantly extend your system configuration to reach advanced goals to satisfy your business needs.

The first goal of workload management is to keep your system in the optimal performance range and away from excessive demand. Excessive demand results in work spending more time fighting other work for access to the same resources. As a result, all work takes longer to complete.

To achieve a stable, predictable system, the goal is to maximize the output of the system without overloading its capacity; that is, we want to prevent the inefficient use of the key system resources (for example, CPU, I/O, and memory) in order to maximize the throughput of the system by finding the optimal zone, as depicted in the theoretical example in Figure 2 on page 9.

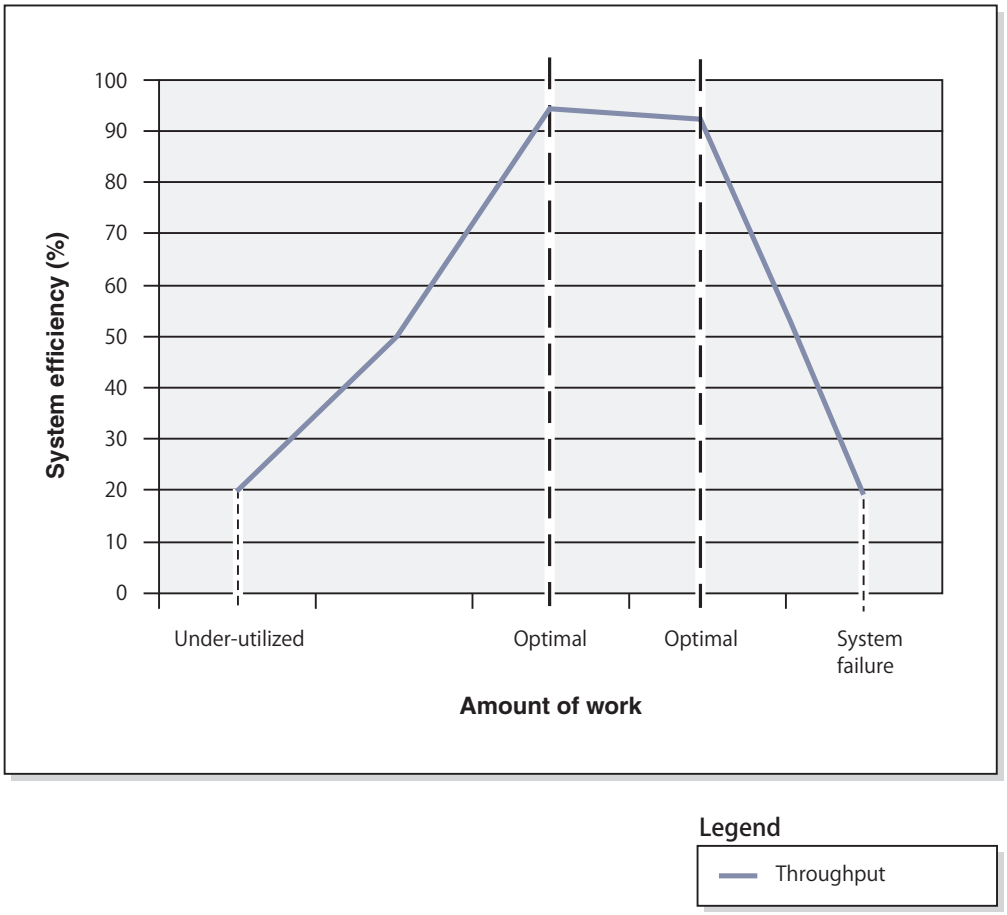


Figure 2. Example of the optimal zone for a system

Since the actual work present on the system, at any one time, varies in terms of its volume and impact on system resources, the demand for one or more of the key system resources (for example, CPU, I/O, memory, and storage) can also be volatile. As soon as demand exceeds supply, the throughput of the system suffers. To make the performance of the database system more predictable, the goal is to level out the peaks in resource demand to avoid reaching resource limits. To smooth out the use of system resources, it is necessary to control the mix of work executing on the system, especially when dealing with work that is more demanding, or larger, than other work. Larger work has much more impact than smaller work. For example, one complex statement on some systems can consume more resources than 10 medium statements.

The most effective action that can be taken, in terms of controlling overall resource consumption, is to control when work is allowed to start execution because minimal resources are consumed before execution starts. This *gatekeeper* approach is effective for all key resources, not just CPU. Also, in current database systems, the resource impact of any piece of work is magnified with the addition of parallelism; delaying the start of execution for any SQL statement also delays the start of any parallel processing done for that statement.

Another approach often used to control resources is to slow down, or starve, a piece of work that is executing to allow other work to use those resources. Typically, this is related to CPU access; mechanisms, such as agent priority and AIX® WLM or Linux WLM, control access to the CPU. While effective in some

scenarios, slowing down work sometimes has unexpected (bad) consequences due to secondary effects on the systems. The following consequences can be a result:

- Higher overall memory utilization because more memory is active at the same time
- Extended lock wait for other work waiting on locks held by the slowed work
- More resource consumption in operating system context switching due to the increased number of threads running.

Overall, experience has shown that it is typically better to delay starting new work rather than slow it down.

Signs of a healthy system

In general, there are system characteristics that are optimal for a typical data warehouse in terms of maintaining system responsiveness.

The following are the core system characteristics that can signify a healthy system:

- Run queue length is less than 10
- Overall CPU utilization is around 80-95%, with system CPU usage below 10-20%
  - The target value for system CPU usage varies by platform. For example, the target is less than 10% on Linux and less than 20% on AIX operating systems.
- Memory utilization is below 100% (that is, no paging)
- I/O waits are 10% or less
- System workload is evenly balanced across all members (that is, no skew or uneven resource demands)

Although these characteristics are only guidelines, they offer a set of objectives that are a reasonable starting point for laying down an initial workload management configuration in a data warehouse. In addition, these guidelines are applicable on a machine by machine basis. You must consider that an entire database system might appear healthy when data is aggregated, but one partition might be unhealthy.

Managing system capacity with DB2 workload management

In this simple conceptual example that is illustrated in Figure 3 on page 11, we have determined that running more than 10 large DML statements at any one time is detrimental to the health of the system and we impose a limit on the number of those statements that can run concurrently.

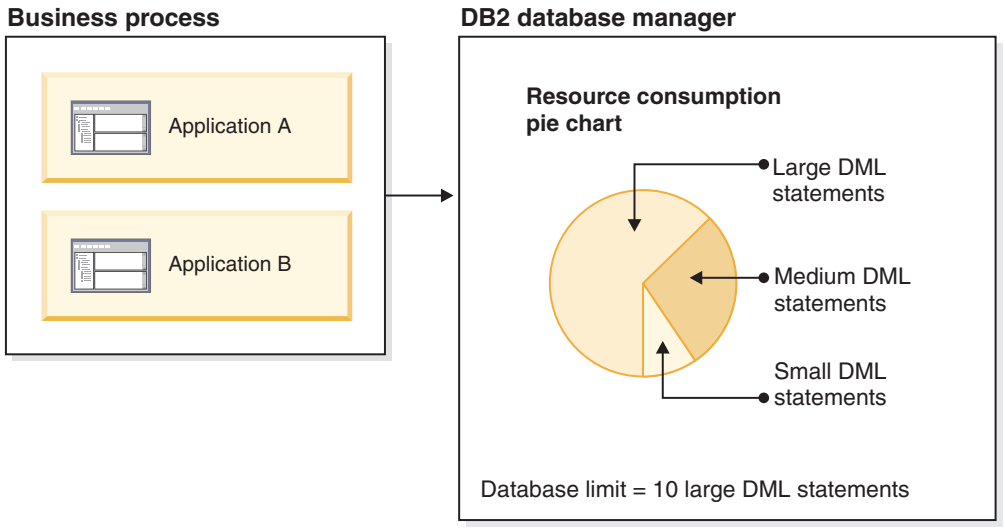


Figure 3. Managing system capacity

The natural location within the DB2 database manager for resource allocations is the DB2 service class. The activity level concurrency threshold (CONCURRENTDBCOORDACTIVITIES) is an effective gatekeeper mechanism. By defining a CONCURRENTDBCOORDACTIVITIES threshold with a limit of 10 on the service class in which the large queries run, we can ensure that no more than 10 queries start executing at any one time, protecting the system.

Concurrency thresholds do not rule out the use of other techniques. For example, AIX WLM can be used to cap the CPU used by low priority work as a whole, meanwhile a concurrency threshold on this same work controls its I/O and memory impact on the system. As an additional benefit, work in the capped service class runs more efficiently when concurrency controls are also in place because we can then control the demands placed on the (limited) CPU resources that are made available to that service class.

Figure 4 on page 12 shows an implementation which uses a work action set to isolate the large DML statements into a separate service subclass and then imposes a concurrency threshold of 10 on that subclass.

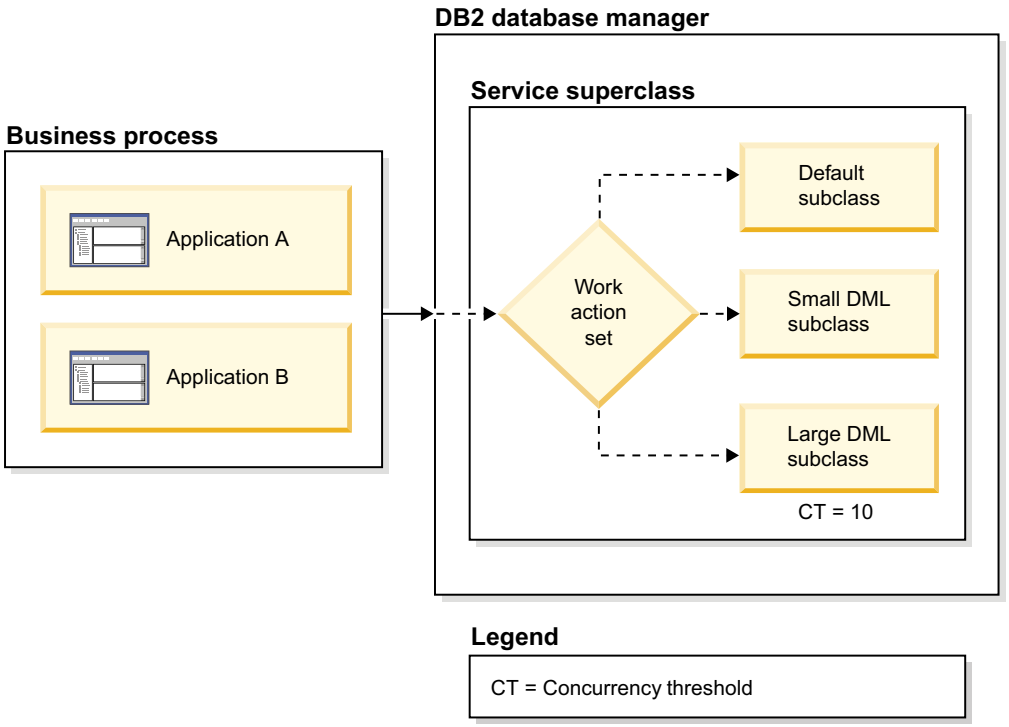


Figure 4. Managing system capacity with DB2 workload management

For a real-world example, Figure 5 on page 13 shows the actual results achieved at a large retail customer that used the same type of approach as used in the preceding simple example. As one can see, the system became more responsive across almost all types of queries as fewer statements were allowed to execute concurrently. The only exception is the *tuned WLM* case for the 70,000 - 100,000 timerons estimated cost class shows that its response time degraded. This case was due to an explicit decision by the customer to move resources elsewhere because slower response times in that category were deemed to be acceptable when compared to the improvements in the other areas that those resources provided. In this case, an important point about workload management is illustrated: workload management does not magically make everything run faster. Instead, with workload management, you get to choose the priorities of the system and directly control the trade-offs made between different work executing on the system.

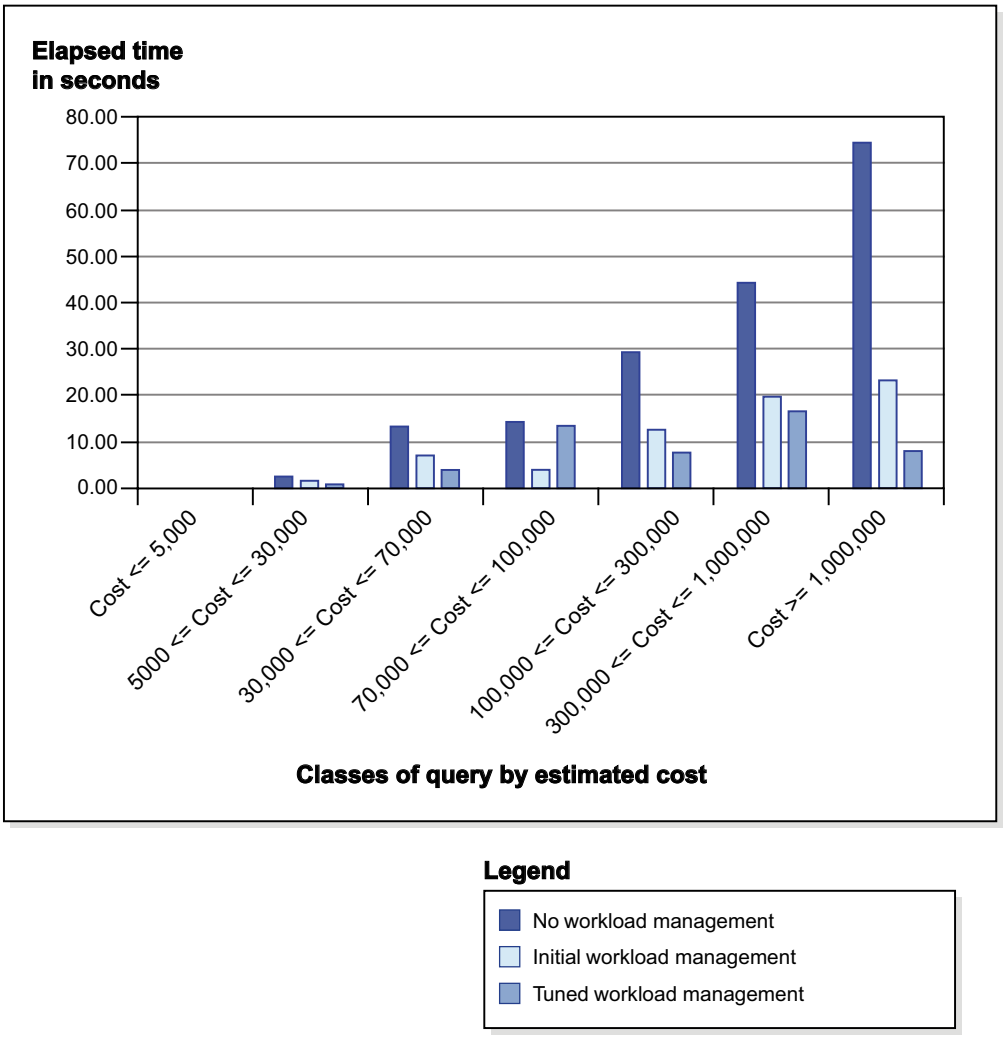


Figure 5. Real-world results from a large retail data warehouse

Design: Configuring DB2 workload management in stages

Why stages?

This best practices paper identifies and defines four stages in the evolution of a DB2 workload management configuration (see the following table) that are very helpful in framing discussions.

Table 1. Stages of DB2 workload management configuration

Stage	Name	Objective
0	Default workload management configuration	
1	Untuned workload management configuration	Learning about your system
2	Tuned workload management configuration	Stabilizing your system
3	Advanced workload management configuration	Dealing with unique aspects

It is recommended that all DB2 for Linux, UNIX, and Windows customers with a data warehouse implement a stage 2 workload management configuration at a minimum. Many of you will likely progress to a stage 3 implementation for your business to address unique performance requirements, or strategic requirements, or both.

The following sections provide a more detailed description of each stage in this new taxonomy.

Stage 0: Default DB2 workload management configuration

A stage 0 DB2 workload management configuration consists of the default workload management configuration established when a database is first created in DB2 for Linux, UNIX, and Windows for both Versions 9.5 and 9.7.

Figure 6 on page 16 shows the default DB2 execution environment for such a database. This configuration has no immediate objective other than to separate user and system work for monitoring purposes.

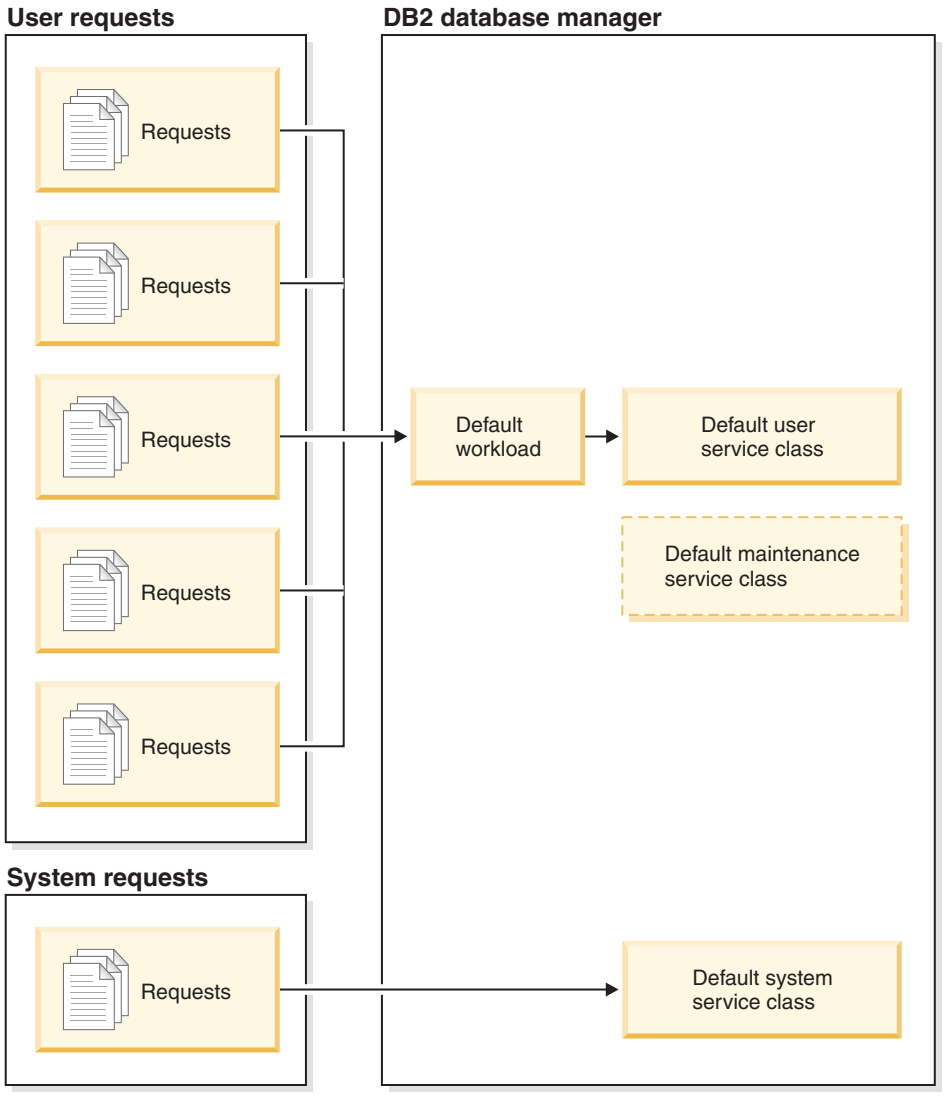


Figure 6. Default DB2 workload management configuration

Within this default configuration, all user connections to a DB2 database are mapped to a DB2 workload. A default workload definition, SYSDEFAULTUSERWORKLOAD, is provided for this purpose. This workload guides all incoming work for the database to be executed in the default user service class, SYSDEFAULTUSERCLASS. All system work executes in the default system service class, SYSDEFAULTSYSTEMCLASS. All background maintenance work, initiated by the DB2 database manager, runs in the SYSDEFAULTMAINTENANCECLASS service class.

Stage 1: Untuned best practices workload management configuration

A stage 1 untuned best practices workload management configuration is the result of transforming the default stage 0 workload management configuration with the creation of a new service superclass and six subclasses. You can do this transformation by applying the template configuration described as part of this best practices paper. All work is mapped into their respective service subclass, based on their estimated cost and type of activity, using a DB2 work class set.

The objectives of this stage 1 untuned workload management configuration are the following:

1. To focus on core database system stability by ensuring incoming demand does not exceed capacity, based on rough guidelines from field experience
2. To gain a better understanding of the nature of the business work being executed on the database and its timing through baseline monitoring. This feedback is used to help tune the template settings to better reflect the actual environment.

The stage 1 untuned workload management configuration consists of a template approach that was created from the key elements identified and experiences gained from successful implementations in a number of customer environments. The basic framework consists of identifying different classes of incoming work, placing controls on resource consumption by the larger pieces, and detecting outliers.

Best practices configuration template

The best practices configuration template is designed to provide a structure that functions well for all warehouse environments and consists of a series of predefined service classes and thresholds. It is intended to help stabilize the performance characteristics of the warehouse (that is, achieving a stage 2 tuned workload management configuration) while providing a solid foundation for additional customization (that is, a stage 3 workload management configuration).

The best practices template uses the estimated cost of each SQL statement to help classify it. If your workload is one where the estimated cost of the work does not correlate, even roughly, with the relative size of the impact of the work on the system, then the template approach espoused in this document will not work for you. To use this template with such a workload, you must use the available reoptimization techniques to align estimated costs with impact. For additional information about this reoptimization technique, see: “Queries with similar estimated costs” on page 96.

If you are unable to use such techniques for your workload, then consider using an alternative approach to workload management. Alternative approaches include using concurrency control for the work on the database system as a whole, regardless of the expected impact, or the priority aging technique, discussed in the DB2 Information Center, perhaps supplemented with AIX WLM hard limits on CPU resource usage.

Template description

The best practices workload management configuration template consists of a single DB2 service superclass that contains a series of DB2 service subclasses. Each service subclass represents a distinct type of work that is usually found in a typical DB2 warehouse environment.

The distinct groups of identified work consist of the following:

- ETL activities (for example, LOAD)
- Five different classes of DML statement queries:
  - Trivial
  - Minor
  - Simple
  - Medium
  - Complex

The template defines a DB2 work class set with a unique work class definition representing each of the preceding groupings. It also defines and associates a DB2 work action set based on these work class definitions on the new service superclass to map incoming work into the appropriate service subclasses that matches its grouping. All other work is left to execute in the default service subclass of the new service superclass and includes the following items:

- DDL and DCL statements
- CALL statements (but not the DML within them)
- Miscellaneous database requests: PREPARE, BIND, DESCRIBE, and others
- Other utilities: REORG, RUNSTATS, and others (that is, all utilities except LOAD)

The deployment of the initial template configuration modifies the SYSDEFAULTUSERWORKLOAD default user workload provided with the DB2 database manager, such that it now guides all incoming work to point to the new DB2 service superclass.

Figure 7 depicts an overall schematic of the template configuration.

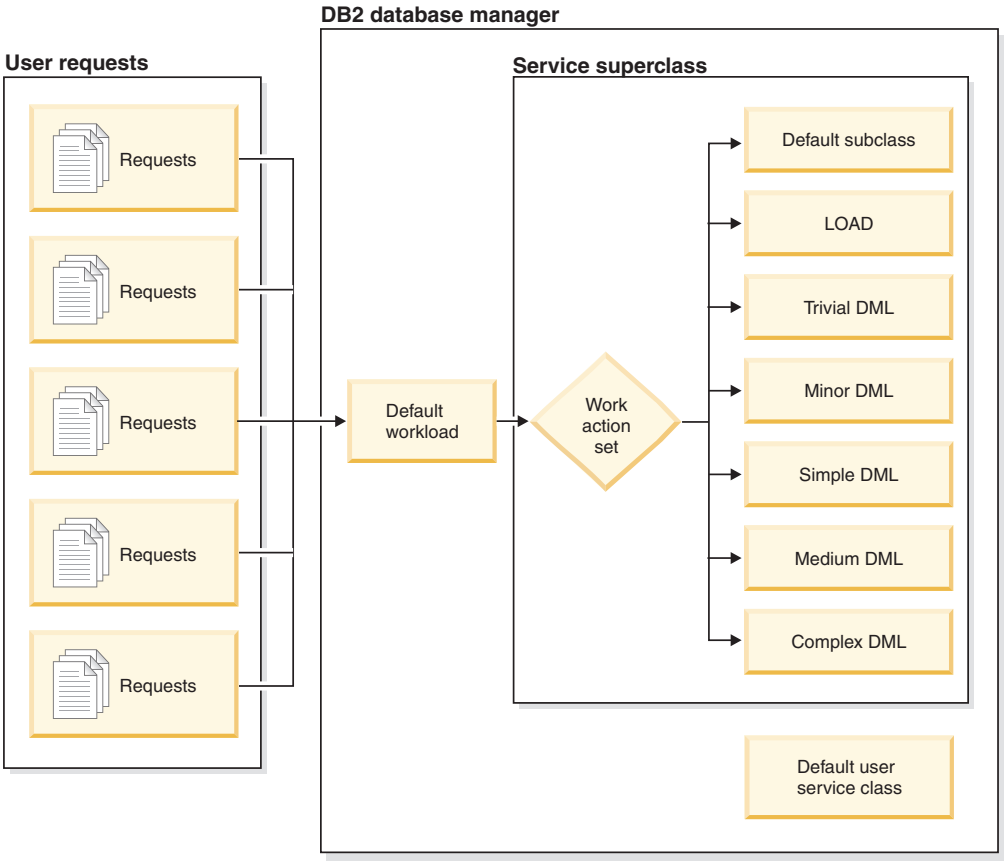


Figure 7. Template for the stage 1 untuned workload management configuration

The template configuration also contains a number of DB2 threshold definitions for both activity timeout and concurrency thresholds. The concurrency thresholds are created in a disabled state in the sample DDL statements provided as part of the best practices template and are activated during the implementation process. The

activity timeout thresholds are enabled by the template, but only to collect information about any violations; they do not stop any activities that exceed the defined limit.

Template work class definitions

A number of work class definitions are provided as part of the best practices configuration template.

The following table shows the expected execution time for the different classes as well as the range of estimated cost values (in timerons) used to identify the different classes of DML statements. Experience in the field has shown that these values serve as a good general starting point for the best practices workload management configuration. A later section describes how to adjust these values to be more appropriate for your system.

Table 2. Template work class definitions

Work class	Expected execution time	Estimated cost: Bottom timeron value	Estimated cost: Top timeron value
LOAD	Unknown	N/A	N/A
TRIVIAL_COST_DML	< 1 second	0	5000
MINOR_COST_DML	< 60 seconds	5000	30,000
SIMPLE_COST_DML	< 5 minutes	30,000	300,000
MEDIUM_COST_DML	< 1 hour	300,000	5,000,000
COMPLEX_COST_DML	> 1 hour	5,000,000	Unbounded

Template threshold definitions

A number of predefined DB2 thresholds are provided with the best practices template.

The ACTIVITYTOTALTIME threshold is used to identify any activity that is running longer than the specified period of time. Such activities might be misclassified or misbehaving work on the database. The following ACTIVITYTOTALTIME activity thresholds are enabled by default in the template:

Table 3. Template activity threshold definitions

Threshold name	Domain (service subclass)	ACTIVITYTOTALTIME threshold criteria	Threshold action
WLMBP_TRIVIAL_DML_ TIMEOUT	TRIVIAL_DML	1 minute	Collect activity data & continue
WLMBP_MINOR_DML_ TIMEOUT	MINOR_DML	1 minute	Collect activity data & continue
WLMBP_SIMPLE_DML_ TIMEOUT	SIMPLE_DML	5 minutes	Collect activity data & continue
WLMBP_MEDIUM_DML_ TIMEOUT	MEDIUM_DML	60 minutes	Collect activity data & continue
WLMBP_COMPLEX_DML_ TIMEOUT	COMPLEX_DML	240 minutes (4 hours)	Collect activity data & continue

The following CONCURRENTDBCOORDACTIVITIES concurrency thresholds are disabled by default in the template:

Table 4. Template concurrency threshold definitions

Threshold name	Domain (service subclass)	Concurrency limit	Queued activity limit	Threshold action
WLMBP_ETL_ CONCURRENCY	ETL (LOAD)	4	Unbounded	None[1]
WLMBP_MINOR_DML_ CONCURRENCY	MINOR_DML	40	Unbounded	None[1]
WLMBP_SIMPLE_DML_ CONCURRENCY	SIMPLE_DML	16	Unbounded	None[1]
WLMBP_MEDIUM_DML_ CONCURRENCY	MEDIUM_DML	8	Unbounded	None[1]
WLMBP_COMPLEX_DML_ CONCURRENCY	COMPLEX_DML	4	Unbounded	None[1]

[1] With the UNBOUNDED queued activity limit, this threshold can never be violated.

These definitions are created in the disabled state as they must be modified to fit both the capacity of the system on which the database warehouse is running and also the actual population of the different groups of work running on the system. This modification is described in the next section.

Stage 2: Tuned best practices workload management configuration

A stage 2 tuned best practices workload management configuration is the result of tuning and customizing the stage 1 untuned best practices configuration. Tuning and customization is accomplished by adding active concurrency thresholds to more closely model the actual working environment. A stage 2 configuration also includes establishing a monitoring regime and is a stable configuration, subject to normal periodic reviews as discussed in a later section.

The objectives of this stage 2 tuned best practices configuration are the following:

1. To achieve a more optimal implementation of a stable, predictable system
2. To identify and deal with any misclassified or misbehaving work in the database
3. To identify the primary sources of work, with accompanying DB2 workload definitions, to allow more enhanced monitoring of performance and resource consumption at the level of the individual sources. This objective is a precursor to further stage 3 customization.

In the example of a stage 2 configuration shown in Figure 8 on page 21, a number of changes have been made to the original best practices template configuration as a result of the process of making the transition from a stage 1 configuration to a stage 2 configuration. A number of customized individual DB2 workloads have been added to separate sets of connections from each other for monitoring and control purposes. The default DB2 workload has been prevented from submitting any work to the system so that any connections mapped to it return an error when work is submitted. All of the workloads still point to the standard service superclass introduced as part of the stage 1 configuration, but the original template subclasses have been modified and unneeded ones have been eliminated. Finally,

two concurrency thresholds have been enabled, one on the Complex DML service subclass and the other on the LOAD service subclass to limit the amount of resources that they can consume.

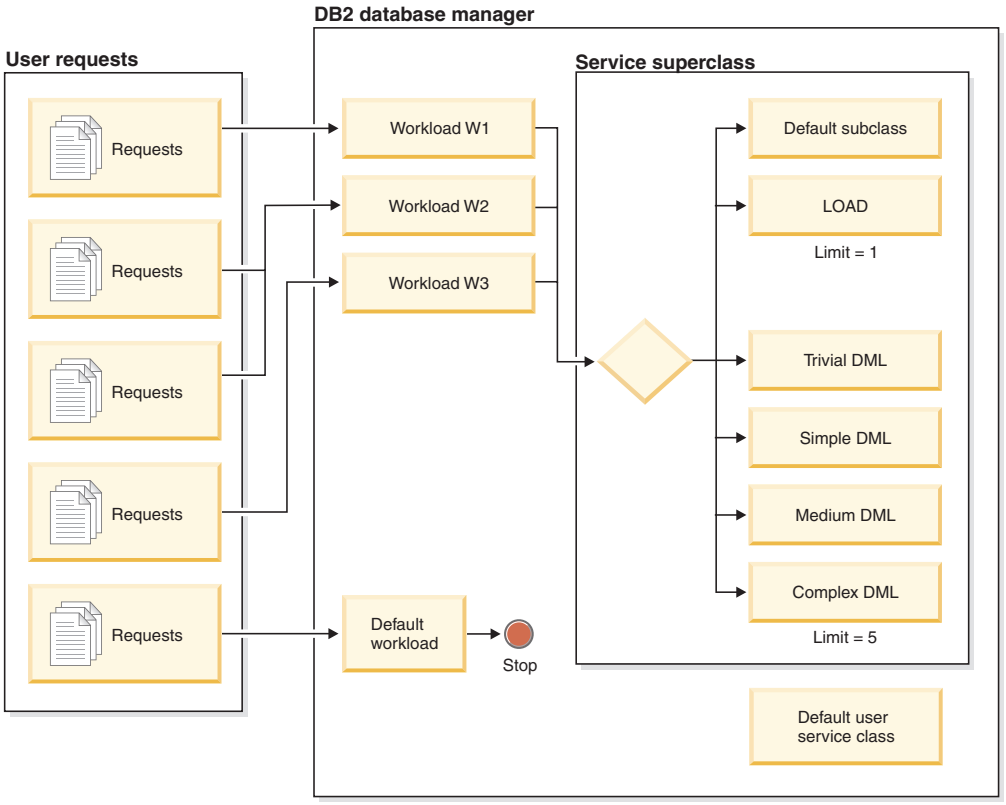


Figure 8. Stage 2: Tuned best practices workload management configuration

Stage 3: Advanced workload management configurations

A stage 3 workload management configuration is any configuration that exceeds or differs significantly from a stage 2 configuration.

Typically, these extensions to a stage 2 configuration are required when dealing with unique requirements that go beyond basic system stability. Examples of such requirements include the following:

- Guaranteeing consistent throughput for one application above all others by introducing the use of AIX WLM hard limits
- Offering different levels of service to end users depending on their funding arrangements

For detailed information about some common stage 3 scenarios and suggested best practices configurations, see: "Advanced configurations: Stage 3 scenarios" on page 55.

Implementation timeline

One last aspect that must be discussed, as part of the new taxonomy for describing DB2 workload management configurations, is the expected timeline for implementation. The actual elapsed time depends on the effort and focus put on workload management, as well as the complexity of the environment being managed and business requirements being imposed upon it.

Although this discussion describes one possible timeline that can be followed, it is not, by any means, meant to be an authoritative example. Many of the steps for advancing from one stage to the next can be condensed or extended as needed. The timeline, depicted in Figure 9, is presented as an example of how you might expect the steps and tasks involved to flow over time.

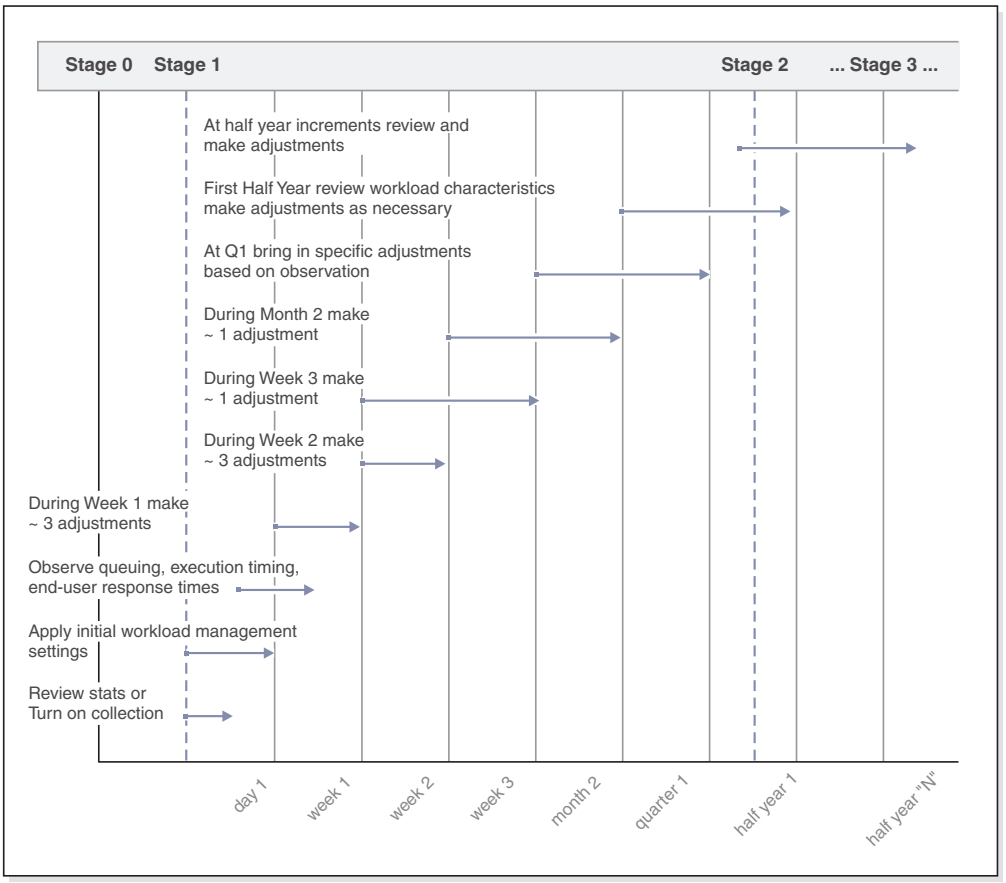


Figure 9. Implementation timeline

Every database starts at a stage 0 configuration as soon as you install or upgrade to a release of DB2 for Linux, UNIX, and Windows that is at Version 9.5 or later. At some point, the best practices workload management template configuration is applied to reach a stage 1 configuration. This stage 1 configuration is followed by a time of monitoring and customization efforts to fine-tune that configuration to reach a comfortable stage 2 configuration. This effort can proceed rapidly at first, but the finer adjustments might need to wait for the demand from longer business cycles and infrequent workloads to be observed before reaching completion (for example, monthly or quarterly reports). If nothing further is needed from the workload management template configuration, a steady-state condition at stage 2 exists in which ongoing monitoring can proceed with suggested biannual checkups

to make sure that no further adjustments are needed. More advanced stage 3 configurations can be explored to address unresolved or new requirements.

Implementation: Reaching a stage 2 workload management configuration

This section focuses on the step by step actions needed to move a database from a stage 0 default workload management configuration to a fully customized stage 2 configuration. This effort is presented as two distinct phases for the sake of clarity. The first phase deals with the steps needed to reach a stage 1 configuration by laying down the best practices template configuration. The second phase addresses the steps needed to reach a stage 2 configuration by customizing the template to better reflect the environment in which it exists.

The following is an overview of the major activities required to take the workload management configuration in a database from a stage 0 configuration to a stage 2 configuration:

1. Apply the best practices workload management configuration template to your system (see: "Template description" on page 17).
2. Collect baseline monitoring information for estimated cost distribution for SQL statements on your system.
3. Adjust template work class definitions (and service classes if needed) to better reflect the real types and distribution of work in your environment.
4. Collect baseline monitoring information for resource consumption by service subclass on your system.
5. Adjust the concurrency thresholds defined on the different service subclasses to better reflect the resource allocation that you want for each one.
6. Define activity thresholds to detect misclassified or misbehaving queries and prevent them from threatening the system.
7. Establish a monitoring regime to ensure the ongoing fitness of your final stage 2 configuration for your system.

   After the best practices workload management configuration template has been customized for your environment, continue monitoring on an ongoing basis with activities 1 through 6 being repeated as needed to keep the configuration tuned with respect to any changes in the environment.

Transition from stage 0 to stage 1

This transition phase covers the steps needed to move from a stage 0 configuration (default DB2 workload management environment in DB2 for Linux, UNIX, and Windows Version 9.7) to a stage 1 best practices template configuration with active concurrency thresholds in place to control the mix of work executing on the database.

To transition from stage 0 to stage 1, follow these steps:

1. **Data collection step:** Enable the collection of request and activity metrics within the DB2 database manager by setting the **mon_req_metrics** and **mon_act_metrics** database configuration parameters to BASE or higher.
2. **Implementation step:** If you are using the IBM Optim Performance Manager (Optim PM) 4.1.1 or earlier, turn off the Optim PM collection of workload management statistics. As a result, the Optim PM will not contain these statistics within its repository for this period and the statistics will not be available for display.
3. **Data collection step:** Create the following event monitors with all event groups defined and use a table space available on all members (you must create a table space across all members, if one does not exist):
   - Activity
   - Statistics
   - Threshold violations

   Note: For your convenience in creating these event monitors, you can use the sample DDL scripts that are provided in Appendix B, "Creating prerequisite event monitors," on page 73.
4. **Implementation step:** Activate the activity event monitor by issuing the following example command:

   ```
   SET EVENT MONITOR DB2ACTIVITIES STATE 1
   ```

   Activate the statistics event monitor by issuing the following example command:

   ```
   SET EVENT MONITOR DB2STATISTICS STATE 1
   ```

   Activate the threshold violations event monitor by issuing the following example command:

   ```
   SET EVENT MONITOR DB2THRESHOLDS STATE 1
   ```
5. **Implementation step:** Apply the best practices workload management template configuration by executing the DDL script, provided in Appendix C, "DDL scripts for transitioning from stage 0 to stage 1," on page 75, on the database.
6. **Analysis step:** Calculate the recommended initial concurrency values for your system. For initial guidance, see: Table 4 on page 20 and "Guidelines for determining initial concurrency threshold values for step 6" on page 27.
7. **Implementation step:** Alter the concurrency thresholds to reflect the calculated values.
8. **Implementation step:** Alter the concurrency thresholds to enable them.

   Note: If you are not comfortable enabling all of the thresholds at once, you can choose to enable the concurrency thresholds one at a time.

   Enabling concurrency thresholds can change the current response time characteristics of work by causing them to wait if sufficient numbers are mapped to a subclass with a concurrency threshold applied. The application of thresholds is intended to bring the mix of work and related resource consumption into the wanted alignment without regard for the response time characteristics of that work.

   If you choose to enable the concurrency thresholds one at a time, start by enabling the threshold for the COMPLEX_DML service subclass first, followed by enabling the threshold for the MEDIUM_DML service subclass, and so on. Issuing the following example statement to enable a concurrency threshold:

   ```
   ALTER THRESHOLD threshold-name ENABLE
   ```

   If the impact of the enabled threshold is too disruptive, you can disable the threshold by issuing the following example statement:

```
ALTER THRESHOLD threshold-name DISABLE
```

Note: Disabling a concurrency threshold results in the slow release of any existing queued activities as currently executing activities complete. New activities entering the system will bypass the threshold completely. If there are queued activities in the threshold, you can release them all at the same time before disabling the threshold by first issuing the following example statement where *x* is equal to a value greater than the current number of executing activities and the number of queued activities:

```
ALTER THRESHOLD threshold-name WHEN
   CONCURRENTDBCOORDACTIVITIES > x PERFORM ACTION
```

9. **Data collection step:** To begin the collection of data by the statistics event monitor, enable automatic collection of the workload management statistics every 60 minutes by setting the value of the **wlm_collect_int** database configuration parameter to 60. The suggested collection duration of 60 minutes is not a strict rule. Longer or shorter collection times are also acceptable.

After completing these steps, the database system will now have an active stage 1 workload management configuration in place. Although the stage 1 workload management configuration is not tuned for the uniqueness of your environment, if implemented with the template concurrency thresholds active, it already helps to ensure that the database system remains relatively stable throughout any variation in workload.

Note: The one caveat to the previous statement about system stability is that the existence of many misclassified activities can still endanger system stability at this point because the activity thresholds allow the misclassified activities to continue execution even after they are detected. This issue is addressed during the transition to a stage 2 configuration.

This database can now be monitored for the information required to make the necessary adjustments to reach a stage 2 workload management configuration.

Guidelines for determining initial concurrency threshold values for step 6

The objective of the concurrency thresholds applied to some of the service subclasses in the best practices template configuration is to constrain the amount of resource consumed by the work in that class. The concurrency thresholds used in the best practices template are evaluated immediately before the execution of individual activities (for example, SQL statements and LOAD) that enter each service subclass. As such, the thresholds decide when a statement can begin execution and thus, begin consuming system resources. Such thresholds are sometimes referred to as *gatekeeper* thresholds in that they control the start of execution, but not subsequent behaviors of an activity after it has started execution.

Obviously, the amount of resource available and the types of work competing for that resource vary with each production environment. With a generic template such as the one described in this paper, it is necessary to make certain assumptions and to provide guidance on how to adjust those assumptions to produce concurrency threshold values more appropriate for the target environment.

This section provides guidance on how to modify the assumptions underlying the default concurrency values, as shown in the table of template concurrency threshold definitions, in such a way that the resultant concurrency values are better suited for the target environment.

There are two elements that can be adjusted to produce more suitable values:

- The total amount of concurrency allowed on the system (the *capacity*, from a concurrency perspective)
- The amount of concurrency to be allocated to each different type of work (the share of the overall capacity that is given to each query group)

In order to provide a simple way to determine a set of initial concurrency threshold values that are appropriate to the underlying system, this document introduces the concept of calculating a theoretical concurrency limit, or *capacity*, for a given system. Then, using a concurrency budget approach, you allocate portions of that limit across the different subclasses defined in the template. The relatively simple techniques outlined here are intended to allow for a reasonable set of initial concurrency threshold values to be set during the process of transitioning to a stage 1 configuration. These initial values can then be better tuned based on actual resource consumption patterns in each service subclass during the transition to a stage 2 configuration.

One last important note to make about concurrency thresholds is that they do not each control the same amount of resource consumption. The individual concurrency units, or *tickets*, each represent one activity execution of the type of work in the service subclass on which the concurrency threshold is defined. The resource impact of these tickets can vary immensely. To further illustrate this important point, let us consider the following example: an activity in the TRIVIAL_DML service subclass might read only one row by using one DB2 agent during its entire execution, while an activity in the COMPLEX_DML service subclass could process millions of rows by using dozens of DB2 agents. The net impact of letting one more activity run in the COMPLEX_DML service subclass is far larger than letting one more activity run in the TRIVIAL_DML subclass. A more generic way of stating this relationship is to say that one activity in the TRIVIAL_DML service subclass has only a portion of the impact on resources that one activity in the MEDIUM_DML service subclass has, and an even smaller portion of the impact of one activity in the COMPLEX_DML service subclass.

Thus, adjustments of concurrency levels across the different service subclasses must account for this significant difference in resource impact by the individual activities. Effectively, this means that in order to allow one more complex query to run concurrently, it is necessary to reduce the concurrency levels of the other classes by more than one query.

The best practices recommendation for setting the initial values is to follow the concurrency budget approach described in the following section and leave the finer adjustments until the later part of the tuning methodology. At that time, more discussion is provided on how to work with consideration for this disproportional impact (see: "Adjusting concurrency threshold values" on page 35).

Determining capacity

As mentioned previously, the first step in identifying a good initial set of values for the concurrency thresholds that are defined on the template service subclasses is to determine a hypothetical overall limit, or *capacity* for concurrency.

Determining the overall capacity, from a concurrency perspective, is a somewhat subjective process and varies by platform and operating system. Our experience has shown that a good, rough estimate of the concurrency levels appropriate for any machine is a multiple between 10 – 12 times the number of cores available to the DB2 database on a data module (as defined in an IBM Smart Analytics System). That is, the best initial estimate of the overall concurrency level suited for the entire database is based on the number of cores available on the physical machine that is dedicated to the database for data storage, regardless of the number of database partitions on that machine.

This initial concurrency level estimate can be best explained by working through the following example: the 7700 IBM Smart Analytics System has a standard data module with 16 cores which is then divided across 8 database partitions, each partition with 2 cores for their dedicated use. Using the estimation technique and a conservative 10x factor, the deemed capacity of the database as a whole is 160 concurrent activities (16 cores x 10 = 160). As long as the I/O demands on the database system are balanced across all members (that is, there is no data skew), then this formula has proven successful at predicting a good initial concurrency limit to use for allocation across service classes.

The important thing to note here is that the suggested estimation technique takes advantage of an observed indirect relationship between a good concurrency level for a database as a whole and the physical attributes for one component of that system. It is not an actual statement of how many things each processor can handle or how many items will be on the system when this concurrency level is used. You must keep in mind that the concurrency thresholds, used in the template, operate as the *gatekeeper* for new activities entering the system and they determine when the application request for a SQL statement (or LOAD) is allowed to start executing. The actual concurrency seen by the operating system, in terms of DB2 agents (that is, operating system threads) executing on behalf of that single activity, and the pattern of low-level processing requests made to the system, can vary greatly depending on the complexity of the activity after it starts to execute. The actual concurrency and pattern of processing requests will often exceed a 1:1 ratio, with the parent activity controlled by the concurrency threshold (for example, letting 1 statement start executing often causes >1 concurrent DB2 agents to begin processing on the system).

The values given for the concurrency thresholds in the supplied DDL script (see: Appendix C, "DDL scripts for transitioning from stage 0 to stage 1," on page 75) are based on using a default 10x multiplier for an 8 core data module. Our rationale behind choosing this approach was that it was acceptable to under-utilize any given system with the initial template, but not to allow over-utilization. The following table gives a quick reference for the different IBM Smart Analytics System models that are offered at the time of publication of this best practices paper. Note that the 5600 model has a lower multiplier to reflect the lower performance characteristics of that model.

Table 5. Lookup table for IBM Smart Analytics System capacity

IBM Smart Analytics System model	Number of cores per physical data module	Initial multiplier	Initial capacity estimate (theoretical limit = number of cores x multiplier)
5600	6 x 2.93GHz (x3650 M2)	5x	30

Table 5. Lookup table for IBM Smart Analytics System capacity (continued)

IBM Smart Analytics System model	Number of cores per physical data module	Initial multiplier	Initial capacity estimate (theoretical limit = number of cores x multiplier)
5600 S	8 x 2.93GHz (x3650 M2)	5x	40
7600 R1.2	4 x 5.0 GHz (Power550)	10x	40
7600 R1.2 Upgraded	8 x 5.0 GHz (Power550)	10x	80
7700 R1.1	16 x 3.55 GHz (Power740)	10x	160

Deciding on whether to use a 10x, 11x, 12x, or higher multiplier requires a judgement call that takes into account the operating characteristics of the system while the full workload is running, and then deciding if more work can be executed on the system. Keeping in mind that the objective for a healthy system is CPU utilization between 85-100% (with under 10-20% of that devoted to system use), you can use the following guideline that explains how to adjust the multiplier after the initial default 10x multiplier has been used to set the concurrency threshold level.

As a guideline, evaluate peak CPU utilization, over the course of a day or more, for a specific data module with active concurrency thresholds in place. Use the following suggested multipliers, depending on the observed peak CPU utilization level:

- If peak CPU utilization is under 65%, consider using 12x as the multiplier
- If peak CPU utilization is between 65% and 85%, consider using 11x as the multiplier

Again, it must be emphasized that changing the multiplier affects not just CPU utilization, but also memory and I/O utilization. Thus, all aspects of performance need to be considered as the multiplier value is changed. If the performance of the individual activities degrades or other resources are exhausted when the multiplier is increased, it is recommended to revert to the previous multiplier value and make smaller adjustments, if further tuning is still desired.

Allocating capacity

After you have established the theoretical limit on overall concurrency for a given database system based on the techniques described in the previous section, the next exercise is to allocate that capacity across the different service subclasses defined as part of the best practices template workload management configuration.

The best practice technique used is a concurrency budget in which a percentage of the overall capacity is assigned to each specific subclass and the relevant concurrency threshold, if any, is set to reflect that portion of the overall limit. The best practices template configuration uses the following concurrency budget to allocate the overall concurrency limit of 80, assumed for the template service class definitions.

Table 6. Template budget allocation for concurrency

Service subclass name	Concurrency budget allocation	Initial concurrency values (10x on 8 cores)	Individual activity resource impact
Default	Shared 10%	8 (shared)	Negligible
ETL	5%	4	Moderate
TRIVIAL_DML	Shared 10%	8 (shared)	Negligible
MINOR_DML	50%	40	Minimal
SIMPLE_DML	20%	16	Light
MEDIUM_DML	10%	8	Moderate
COMPLEX_DML	5%	4	Heavy

At this point in the overall process, it is recommended that the concurrency budget is used as presented. There will be an opportunity to adjust the concurrency threshold values at a later point based on the actual resource consumption by the work in each service subclass.

There are several things of interest to be pointed out with this budget approach:

- Although they are not themselves controlled by concurrency limits, the impact of the work being processed in the Default and TRIVIAL_DML service subclasses must still be acknowledged and accounted for in the budget. These two subclasses have been allocated to share 10% of the overall capacity to ensure that the resources consumed by that work are represented.
- The ETL service subclass was assigned 5% and has a concurrency of four loads that are allowed to be executing at any one time. Be reminded that this concurrency value is rather arbitrary because the use of the load utility varies greatly among customers and the load utility can be used for different purposes, such as loading staging tables versus loading base tables. When adjusting the concurrency value, be aware that experience has shown that executing more than eight loads at a time often leads to degradation in the performance of the individual loads (they all slow down). At some point, this performance degradation outweighs the time spent waiting on the concurrency threshold and it is better to not let more loads run concurrently.
- The MINOR_DML service subclass has been allocated a large percentage of the concurrency limit, despite the fact that the work consists of very small, individual statements. This concurrency limit is not in place to control the average consumption level by work executed in this class, but rather to control the peak usage when a flood of work of this type arrives at the same time. Experience has shown that this class of work typically represents a significant portion of the work asked of a database system and has a high volatility in terms of arrivals, requiring a control to be put in place to handle those moments.
- Complex queries have been allocated 5% of the budget with a concurrency of 4 given as a result. These queries are often the largest ones to hit the system and actually use the most resources per executing item. Experience has shown that very few systems can run well with more than 10 complex queries running on them at the same time.

Transitioning from stage 1 to stage 2

This transition phase covers the steps needed to move from a stage 1 untuned best practices template configuration to a fully customized stage 2 tuned configuration. This transition is done by adjusting the work class definitions and the service classes, as necessary, to better reflect the actual mix of work on the database, and then, if you want, adjusting the resource consumption of each service class by adjusting concurrency threshold values.

To transition from stage 1 to stage 2, follow these steps:

1. **Data collection step:** Gather detailed monitoring information for each service subclass on the distribution of estimated costs.
2. **Analysis and implementation step:** Use this estimated cost distribution to adjust the work class definitions in the DB2 work class set.
3. **Data collection step:** Gather detailed monitoring information for each service subclass on the distribution of resource consumption.
4. **Analysis and implementation step:** Use this resource consumption information to adjust the concurrency threshold values to have real resource consumption better reflect the levels you want.

After completing these steps, the database system now has an active, customized stage 2 configuration that can help ensure that the performance of the database system remains stable and predictable. In addition, the stage 2 configuration provides a solid foundation on which to build a more advanced stage 3 configuration, if that is what you want.

Guidelines for transitioning to a stage 2 configuration

This section provides more detailed information about the steps related to transitioning to a stage 2 configuration.

Guidelines are provided for the following steps:

- Step 1: "Gathering detailed monitoring information for adjusting work class definitions"
- Step 2: "Adjusting work class definitions" on page 34
- Step 3: "Gathering detailed monitoring information for adjusting concurrency threshold values" on page 35
- Step 4: "Adjusting concurrency threshold values" on page 35

Gathering detailed monitoring information for adjusting work class definitions

The first phase in the process of transitioning to a stage 2 configuration is to adjust the work class definitions to better reflect the actual work running on your system.

To make these decisions, it is necessary to collect the range of estimated costs for each service subclass and examine how the values are distributed across the existing range of values defined in the work class set provided with the template (see: Table 2 on page 19).

To get the distribution of actual estimated costs, you can use the activity event monitor[1] to collect the coordinator perspective for all work executed in each target service class by setting the COLLECT ACTIVITY DATA attribute for that service class to COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS. This action causes detailed information to be collected by the activity event monitor for each activity that is executed in the service class. This information provides details about the execution time, origin, statement text (if applicable), and estimated costs, in addition to other items.

Although it is desirable to collect all the SQL statements that are executed, this collection might not be feasible due to time or system resources. The important objective here is to acquire a (very) good representation of the mix and range of work that runs in the service class so that any conclusions drawn from the data are valid for all SQL statements to which they apply.

To examine the distribution of values from data in the activity event monitor, you can use a simple SQL statement to categorize and count the different estimated costs in order to create an overall distribution representing the complete range of estimated costs encountered (see: "Analyzing activity event monitor data" on page 81). This information is then used to decide what adjustments, if any, are needed to the original work class definitions provided in the template.

An alternative source of estimated cost information is the statistics event monitor that was enabled as part of stage 1 baseline monitoring. The information collected within the statistics event monitor represents the set of monitoring information for each defined workload management entity since the last time statistics were collected from each database member. For additional information about using the statistics event monitor alternative, see: "Analyzing activity event monitor data" on page 81.

Note: When using the estimated cost histogram from the statistics event monitor, consider the following points:

- The histogram does not record estimated costs for nested DML statements. Nested DML statements are those statements issued from within a stored procedure (a routine invoked by a CALL statement), or those statements issued from within a UDF (a routine invoked by another DML statement). If the work being analyzed includes such DML statements, then the activity event monitor is the best way to analyze that work, as well as providing insight into what statements are invoked and in what contexts.
- The scale of the individual bins in the histogram data is logarithmic, which means that the range of estimated cost values that appear in the bins to the right are significantly larger than that appearing in bins to the left. This logarithmic scaling needs to be accounted for during the analysis phase. One example of how the logarithmic scale can affect analysis is when many entries appear in a bin giving the impression of a homogenous workload. If that bin is a higher-order bin (one representing a wide range of possible values), then that

1. The following sections primarily refer to the activity and statistics event monitors as sources for the information needed to make tuning decisions. The following document reference is valuable when looking at the different monitor elements: http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.mon.doc/doc/r0007595.html

impression might be false. If most of the work appears in a high-order bin or bins, it is possible to customize a histogram template by using the CREATE HISTOGRAM TEMPLATE statement. By changing the range of values, represented by each bin, in such a way that the population is moved towards the lower-order bins, you can get a better idea of the underlying distribution of the work.

Adjusting work class definitions

This section describes how to analyze the distribution of work in the predefined work classes that are provided as part of the best practices template configuration. The objective here is to help to ensure that the population distribution in any service class is fairly homogeneous and closely grouped with any unused or lightly used work classes merged into the surrounding ones.

After a picture (ideally in the form of a histogram) of the estimated costs distribution has been created, it must be analyzed to see what adjustments, if any, are needed to achieve the goal of having the work in the service subclass represent a closely grouped set of queries in terms of their individual levels of resource consumption.

The goal is to get a relatively homogenous population of queries (resource impacts are of a similar magnitude) in each of the different service subclasses so that the effect of any concurrency threshold, and the effect on other queries, is predictable. If queries are not of the same impact magnitude in each service subclass, the effect of concurrency control can be unpredictable and unstable because system performance would vary with the impact magnitude of the items allowed to execute. In illustration, if a service subclass contained work that had both 1X and 2X impact magnitudes on resources, the system would be less stressed when a 1X item was run and more stressed when a 2X item was run.

More details on the basic techniques that are used during this analysis are outlined in Appendix D, “Techniques for adjusting work class definitions,” on page 81.

Handling ETL work that does not use LOAD

The ETL service subclass was introduced to accommodate the normal data ingest processing that occurs in a data warehouse. In the best practices template, any LOAD invocation is identified and mapped to the ETL service subclass on the assumption it is part of ETL processing. In some scenarios, this work is done by using methods other than LOAD, such as using INSERT statements, so that work does not end up being mapped to this service subclass.

In such cases, the typical implementation would do one of the following:

- Create a unique workload for the ETL process and define it to submit work directly to the ETL service subclass (not to the WLMBP_MASTER service superclass), bypassing the mapping by the work action set
- Create a unique ETL workload and a new ETL2 service superclass, making the workload direct all the work to the new ETL service superclass. If there is a mixture of different work types (for example, SQL queries and LOAD) used as part of the ETL process, then the same overall methodology and template described in this document can be applied within the new ETL service superclass to monitor and control (if you want) the different types of work.

  This implementation allows the resources for the ETL process to still be tracked and controlled independently of other work. This approach can also be used if the LOAD utility was used by other work outside of the ETL process and there

was a desire to track them separately. In that case, the non-ETL LOAD work would be mapped to the ETL service subclass in the template, while the ETL process work would be in the new, separate ETL2 service superclass.

Gathering detailed monitoring information for adjusting concurrency threshold values

The next phase of the transition process to a stage 2 configuration is to evaluate and, if you want, adjust the levels of resource consumption by each service class in your workload management configuration. It is important that the information for this step is freshly gathered after all work class definition changes have been made. Fresh information is a requirement because the content of each service class, or rather the relative nature of their impact on resources, figures heavily in the overall decision process.

The following two sets of information are required in order to do this work:

- System-level monitoring of the entire data module
- Specific monitoring of each DB2 service class aggregated across all members on that same data module

At this stage, you can choose to look at only a representative data module for this monitoring in order to keep things simple. Use the same data module you used to determine the theoretical system concurrency limit (see: "Guidelines for determining initial concurrency threshold values for step 6" on page 27).

System level monitoring can be achieved by using the **vmstat** command to obtain the system and user utilization of the CPU resources on the target data module. To get the service class perspective on CPU utilization, the recommended approach is to use the statistics event monitor data that is being collected at a regular interval and use the **total_cpu_time** monitor element.

In order to get the system and DB2 monitoring information in a synchronized interval, the most effective way is to schedule a script to run on the target data module (for example, a `cron` job) to collect **vmstat** information every 30 minutes. Start the script to run at the top of the next hour in order to align its results with the time interval used by automatic statistics collection provided by the DB2 database manager[2]. For an example script, see: "Sample E1: Sample script for collecting vmstat data" on page 99.

Adjusting concurrency threshold values

After the work class definitions have been modified to better reflect the actual work running on the database, it is now necessary to review and adjust the concurrency thresholds. Using fresh, new monitoring information from the database, the concurrency threshold levels are adjusted, as needed, to bring actual resources consumption by each service subclass closer to the levels that you want.

The objective here is to get to a satisfactory allocation of system resources, based on actual monitoring metrics, versus the theoretical budget allocation done during the stage 1 transition process steps. The reality of the actual workload needs to be recognized and the levels of resource consumption, specifically the CPU resources in this case, adjusted based on the response times being seen by the different work

2. Starting with DB2 Version 9.7 Fix Pack 1, the workload management statistics collection interval is synchronized relative to a fixed start time (a day of the week and an hour of the day), rather than relative to when the DB2 database was activated.

classes versus the performance expectations for each class. Resource consumption is adjusted by changing the concurrency threshold values used in each service subclass.

One very important aspect to keep in mind when adjusting concurrency values is that the concurrency levels reflect the class of work in the service class that they are imposed upon. That is, concurrency values are relative values, not absolute. Thus, you cannot exchange one concurrency *ticket* from the TRIVIAL_DML service subclass and move it to the COMPLEX_DML service subclass because the relative impact on the system of one additional complex query outweighs that of one less trivial query by many times. The impact on the system, when modifying concurrency threshold values between the different service classes in this best practices configuration, is not proportional.

The following table gives illustrative guidance on how to adjust concurrency levels while keeping the disproportional impact fact in mind. As an example, decreasing the value of the concurrency threshold on the COMPLEX_DML service subclass theoretically allows you to raise the concurrency threshold on the SIMPLE_DML service subclass by 4. Conversely, you need to decrease the concurrency limit of the SIMPLE_DML service subclass by 4 in order to increase the COMPLEX_DML service subclass limit by 1.

Table 7. Relative impacts of adjusting concurrency threshold levels between service subclasses

From (row)/To (column)	MINOR_DML	SIMPLE_DML	MEDIUM_DML	COMPLEX_DML
MINOR_DML	1	0.4	0.2	0.1
SIMPLE_DML	2.5	1	0.5	0.25
MEDIUM_DML	5	2	1	0.5
COMPLEX_DML	10	4	2	1

Again, this table of relative impacts is provided only as a conceptual guide. The actual relative impact depends on the actual workload in each service class. In the end, the overriding guidance is to simply adjust concurrency levels slowly and in small increments while observing the net effects after each change.

The process of evaluating and adjusting concurrency threshold levels, based on the monitoring information gathered, consists of the following steps:

1. **Analysis step:** Using the `vmstat` data, calculate the total consumed CPU time, in seconds, for the target data module by using the following formula:
   ```
   (60 minutes * 60 seconds) * (system CPU utilization + user CPU utilization)
   ```
2. **Analysis step:** By using the statistics information gathered in the statistics event monitor, calculate the total CPU consumed by each service class across all members defined on the target data module in the following way:

```
SELECT SUM(delta of extracted TOTAL_CPU_TIME from
XML_DETAILS column from previous collection interval and this one)
FROM event monitor
WHERE service class is on data module
GROUP BY service class name
```

For an example SQL script, see: "Sample E2: Calculating the CPU consumed per service class" on page 100.

3. **Analysis step:** Calculate the percentage of CPU consumption of each service class, relative to the whole, by using the following formula:

   ```
   (Total CPU time consumed by a service class on target data module
   during the last 60 minutes) / (Total CPU time consumed on target
   data module during the last 60 minutes) * 100
   ```

4. **Analysis step:** Determine which service classes should get more or less CPU resources based on their current performance. For each service subclass which has an active concurrency threshold defined on it, look at its response time metrics as reflected in the lifetime average (check the LIFETIME_AVG output column after running the following sample script: "Sample E2: Calculating the CPU consumed per service class" on page 100). Assuming that there are not any extenuating circumstances such as statements encountering lock waits or I/O waits, then the shown response time reflects the result of the current amount of CPU being consumed. In a service subclass, providing work with access to more CPU might improve its overall performance. However, CPU access is not a panacea for performance woes; make an adjustment incrementally and one service subclass at a time to fully evaluate the impact of the change that was made.
   - If CPU utilization is less than 100% and the work in the service class is experiencing short wait times (check the TOTAL_WAIT_TIME output column after running the following sample script: "Sample E2: Calculating the CPU consumed per service class" on page 100), you can try increasing the CPU resources available to that service subclass until 100% CPU utilization is reached.
   - If 100% CPU utilization has been reached, providing more CPU to one service class requires removing it from another service class, potentially further slowing down that set of work. Determine which service class can have CPU removed from it to supply the other one with more CPU access. CPU access can be adjusted in a 1:1 manner, such that taking 5% of CPU away from the COMPLEX_DML service subclass allows you to directly apply 5% of CPU to the MEDIUM_DML service subclass, and so on.
5. **Analysis step:** Based on the results of the previous step, adjust the amount of CPU consumed by each service subclass by using the following guidelines:
   - If CPU consumption is higher or lower than you want, then check if the concurrency thresholds were influencing the result by checking that the value of the **queue_time_total** monitor element, for the relevant concurrency threshold on the service class for the last 60 minutes, is greater than zero.
     - If the value of the **queue_time_total** monitor element is zero, then the concurrency threshold did not restrict CPU consumption during this interval
     - If the value of the **queue_time_total** monitor element is not zero, then the concurrency threshold is indeed restricting CPU consumption. The value of the **queue_size_top** monitor element for the relevant concurrency threshold gives some indication of the degree of restriction by showing how large the queue was, at its deepest, for the concurrency threshold.
   - If CPU consumption is too high, then you want to reduce concurrency to reduce CPU consumption.

- If a concurrency threshold was not in place on this service subclass, or the existing one was not taking any action (that is, **queue_time_total** = 0), then you can set the threshold value to 1 less than the highest level of observed concurrency (as per the value of the **concurrent_act_top** monitor element seen for this service class) so that the concurrency threshold begins to have an effect.
- If there is an existing concurrency threshold which is taking action (that is, **queue_total_time** > 0), then you can lower that threshold value by 1 or more to reduce CPU consumption. If the concurrency threshold is already set at 1, then you are done (that is, you are out of concurrency options and you have to look at more advanced stage 3 options, such as using hard limits from AIX WLM).

• If CPU consumption is lower than you want, then you are going to increase concurrency to increase CPU consumption.
- If a concurrency threshold was not in place on this service subclass or the existing one was not taking any action (that is, **queue_total_time** = 0), then you are done; the current CPU resource consumption is the true consumption for this service subclass.

  Note: If you want to move the unused CPU capacity, by way of AIX WLM configuration, to another subclass based on this low CPU consumption information, you might want to put a concurrency threshold in place that has a concurrency value which is the same as the value of the **concurrent_act_top** monitor element. This action helps to ensure that future concurrency in this service subclass does not exceed the level currently seen in the monitoring data, helping to ensure that the moved CPU capacity is not requested later by this service subclass.
- If there is an existing concurrency threshold that is taking action (that is, **queue_total_time** > 0), then the concurrency value can be raised to allow more work to begin executing in this service subclass. However, remember that workload management is a *zero-sum* problem, so any increase in concurrency in one place must generally be balanced with a relatively disproportional decrease elsewhere in the configuration to keep the system stable.

6. **Data collection, analysis, and implementation step:** Repeat monitoring and adjustment processes until you are satisfied with the general allocation of resources within the system as a whole.

In addition to CPU resource consumption, it is also possible to use a similar approach to monitoring and adjusting concurrency levels for the consumption of other resources, such as I/O and memory, if these resources are contentious for your specific environment and workload.

Adjust concurrency threshold for the ETL service subclass

The process of adjusting the concurrency threshold value for the ETL service subclass is identical to the one used for the other subclasses.

Final steps to complete a stable stage 2 configuration

After you have completed the transition to a customized stage 2 workload management configuration based on the best practices template presented in this document, you should now have a database system that is more stable and predictable than when you started, with performance that is consistent and acceptable.

What remains to be done is to establish a set of ongoing operational monitoring processes and reports. This monitoring information, collected over time and across different business cycles, provides you with pertinent information about the performance of the system and the ongoing suitability of your workload management configuration to your database workload.

Implementing an ongoing monitoring regime is highly recommended because a database system is a dynamic entity whose behavior and demands can change over time. Every business experiences change due to growth or internal weekly, monthly, and quarterly business cycles of their own which often place different demands on their underlying database systems. Any of these naturally occurring factors could result in requiring further tuning of your workload management configuration. For a more detailed discussion about what aspects to consider for an ongoing monitoring regime, see "Monitoring: Maintaining a stable stage 2 configuration" on page 43.

The following points summarize two other aspects of the workload management configuration that can also be investigated at this point and are discussed in more detail in following sections:

- Putting protective measures in place to detect and prevent misbehaving queries from disrupting the system
- Defining additional DB2 workloads to provide a finer granularity of control and awareness to your configuration

Finally, sometimes there are additional requirements or objectives for workload management that require a configuration that is more advanced than the configuration established for stage 2. These advanced configurations are generically referred to as stage 3 workload management configurations and encompass a wide variety of differences. For initial thoughts and guidance on how your stage 2 configuration can be modified to accommodate a set of the most commonly encountered stage 3 scenarios, see: "Advanced configurations: Stage 3 scenarios" on page 55.

Protective measures

On some systems, there is an occasional SQL statement which varies greatly, in one behavior or another, from the norm established by other SQL statements in the same service class. Such variance can badly affect the response time for an individual statement or the system as a whole. This section discusses some techniques that can be put in place by using DB2 thresholds to help detect these variants (sometimes referred to as *rogue* or *outlaw* queries) and, if necessary, stop them.

A number of activity thresholds were implemented as part of the best practices template configuration to identify and collect details about activities that were behaving abnormally when compared to the expected behaviors for the work in each service class. With the availability of real monitoring data such as that collected during the tuning of the stage 2 workload management configuration, it is now possible to adjust the template activity threshold values to better reflect the realities of the actual workload running against the database.

To do this adjustment, we need to compare the initial activity threshold values of the template to the distribution of values in the monitoring data for each DB2 service class. Similar to the method used to adjust the concurrency threshold values, this work must be done after the adjustments to the estimated cost ranges used for mapping to service subclasses and concurrency threshold values have

occurred and fresh monitoring information has been gathered based on these modifications. The primary reason for this requirement is that the ACTIVITYTOTALTIME threshold encompasses time spent waiting on concurrency thresholds and this time will vary as the workload management configuration is tuned.

Because the template activity thresholds are for the ACTIVITYTOTALTIME threshold, examine the coordinator partition for work in each service class over the period of monitoring to check the average and high water mark values of the lifetime monitor elements. This examination can be done from the activity event monitor data by using the difference between the individual **time_created** and **time_completed** monitor elements for statements whose estimated cost would place it in the same service subclass (see: "Sample D1: Determining estimated cost distribution based on activity event monitor data" on page 82).

From the statistics event monitor data, the same thing can be done by using the **coord_act_lifetime_avg** and the **coord_act_lifetime_top** monitor elements. If the statements being monitored are nested SQL statements, such as those invoked from within stored procedures, then you have to use the activity event monitor approach because these metrics do not reflect nested SQL times. For an example SQL script, see: "Sample E3: Determining the maximum activity lifetime value from statistics event monitor data" on page 106. More detailed distribution information can be found in the CoordActLifetime histogram information gathered for the specific service class over the monitoring period. The following document link is a useful reference for information to understand histograms better: http://www.thekguy.com/histograms.html.

The key objective to keep in mind when adjusting the activity threshold values for ACTIVITYTOTALTIME, or indeed any threshold, is that you want to be informed about only truly abnormal events and not the normal outliers that occur in any processing system. That is, if the range of values seem acceptable in the monitoring information, then the value that you use for the activity threshold must be greater than the highest outlier value (perhaps by a significant amount, such as at least 50% greater). Essentially, you want a value that definitely informs you about the existence of that activity so that you can act on it, such as doing in-depth postmortem analysis, or stopping it, or both.

The final aspect to consider, regarding the activity thresholds that the template defines, is the following question: Must the thresholds continue to be defined as CONTINUE, or can they be altered to implement the STOP EXECUTION action? The choice depends on how disruptive such outliers are to the stability of the system and whether you choose to implement more resource-specific thresholds such as CPUTIME and SQLROWSREAD. Because the ACTIVITYTOTALTIME threshold covers both the active and passive phases of execution of an activity such as waiting on a lock or on a concurrency threshold, it typically is not an effective reflection of the true impact of an activity on the system. As a result, it is generally best to leave the action as CONTINUE and implement more specific thresholds as discussed next.

After the template activity thresholds are modified to reflect the actual work, you can consider adding additional activity thresholds on each service class to catch abnormal or unexpected behavior in other aspects, such as the following (typically, the first two activity thresholds listed here are of interest to most DBAs):

- Excessive CPU consumption by using the CPUTIME threshold (see the **act_cpu_time_top** monitor element)

- Excessive I/O consumption by using the SQLROWSREAD threshold (see the **act_rows_read_top** monitor element)
- Excessive system temporary table space consumption by using the SQLTEMPSPACE and AGGSQLTEMPSPACE thresholds (see the **temp_tablespace_top** and the **agg_temp_tablespace_top** monitor elements[3])
- Excessive network traffic through the SQLROWSRETURNED threshold (see the **rows_returned_top** monitor element)
- Detecting inefficient applications through the use of the CONNECTIONIDLETIME and UOWTOTALTIME thresholds imposed on the service superclass (see the **client_idle_wait_time** and the **uow_total_time_top** monitor elements)

Although the previous discussion assumed that any thresholds being defined would be applied to a specific service subclass, you can decide to apply these thresholds directly to a specific service subclass, the service superclass, or to the database as a whole. The important thing to remember is that the values used in the threshold definition must be derived from monitoring data which reflects the full range of statements to which it will be applied. For example, if you are applying a CPUTIME threshold to the database as a whole, you need to look at the CPU values for all statements running on the database.

If you want to apply such thresholds to only a subset of the work running within the service class, create unique DB2 workload definitions for the subset (as discussed in a following section) and then applying the thresholds directly to that workload rather than on the database as a whole or a specific service class.

A note about using the REMAP ACTIVITY

At the time of publication (contemporary with DB2 Version 9.7 Fix Pack 4), the DB2 workload management best practices do not recommend the use of the REMAP ACTIVITY action that is available for some activity thresholds, such as CPUTIMEINSC and SQLROWSREADINSC. It is recommended to use the STOP EXECUTION action to terminate any disruptive queries.

The rationale for this recommendation is that although the REMAP action does allow you to place activities into the proper service class when they have been misclassified due to an incorrect estimated cost, this action has no effective impact on resource consumption (at this time) when using concurrency thresholds. These thresholds do not react or account for remapped activities when determining concurrency levels. The result is that when an activity is remapped into a new service class with a concurrency threshold, the control on resource consumption, that you want, is not enforced.

Creating workloads

Creating workloads is a separate, but important part of the stage 2 customization process. A finer granularity of control and awareness is introduced into the workload management configuration by identifying the primary sources of work using DB2 workload definitions.

Note: It is also beneficial to create workloads in stage 1, but it is not a prerequisite. You can get an early start on this workload creation phase during the stage 1 configuration, if you want.

3. Note that these monitor elements are only updated by activities that have a temporary table space threshold applied to them.

By creating a DB2 workload for each unique set of connections representing a particular application or business entity, it becomes possible to monitor their impact on the system separate from all others and also impose additional or unique threshold values on activities entering the system from that workload. The values of different connection attributes can be pulled from the execution data previously collected in the activity event monitor and analyzed to see if there are any connections of ongoing interest. For an example SQL script, see: "Sample E4: Viewing connection attributes and DB2 workload assignments from activity event monitor data" on page 107.

To validate what connections are mapped to each workload definition, you can look at the **workload_id** monitor element in the activity event monitor output (for a sample SQL script, see: "Sample E4: Viewing connection attributes and DB2 workload assignments from activity event monitor data" on page 107).

Many administrators define unique workloads covering the vast majority of their business and then use the SYSDEFAULTUSERWORKLOAD default user workload, provided with the DB2 database manager, as a way to identify any unknown or new work. Administrators can then choose to either block such work from entering by disallowing access from the default workload, or, more commonly, collect detailed activity information about this type of work when it occurs. In addition, administrators can tightly restrict resource use, by the work as a whole, by mapping the default user workload back to the original default user service class and applying a concurrency threshold value of 1 to that service class.

It is important to ensure that all new workloads, with the possible exception of SYSDEFAULTUSERWORKLOAD that was discussed earlier, continue to map to the WLMBP_MASTER service superclass introduced as part of the best practices workload management template. This mapping of the new workloads ensures that their work is classified and controlled within the service classes and concurrency thresholds that were introduced by this best practices process.

Important: Workloads are only allowed to be used by users authorized to use them. To allow a new workload to be used by users intended to use it, a user who holds ACCESSCTRL, SECADM, or WLMADM authority must grant the USAGE privilege on that workload to the user, group, or role by using the GRANT USAGE ON WORKLOAD statement. To have all users considered, grant USAGE to the PUBLIC group.

Monitoring: Maintaining a stable stage 2 configuration

Monitoring is a critical part of maintaining a system that has reached a stable stage 2 workload management implementation in order to ensure that the system continues to remain healthy over time. Although there will be other monitoring requirements imposed on the system by the business for accounting and tracking reasons, this section presents some general guidelines and suggestions to develop a good, ongoing monitoring regime to maintain a stable workload management configuration.

There are three fundamental objectives underlying the proposed monitoring regime:

- Watching for signs that the system remains in a healthy state
- Investigating and resolving any individual activities that are identified as disruptive (also referred to as *rogue* or *outlaw* queries)
- Watching for changes in resource consumption patterns (both estimated and actual)

The first objective is met by a simple monitoring of the key attributes that represent a healthy, responsive system to ensure that they all remain in the optimal range over time. For additional information, see: "Monitoring system health."

The second objective is achieved by looking at any output from the DB2 activity threshold definitions created to act as *guardians* over activity behaviors. These guardian activity thresholds identify any activities behaving in an abnormal manner relative to the established norms of their peers in a specific service subclass. For more information, see: "Monitoring activity behaviors" on page 45.

The third objective is satisfied by establishing an ongoing monitoring process which watches both the health of the system as a whole, and the range of estimated costs and resource consumption within each service subclass. The results of this monitoring can be used to identify any overall change trends before they become disruptive to the system. They can also be used to adjust the baseline values used by the activity threshold definitions to identify abnormal activity. For more information, see: "Monitoring system behaviors" on page 47.

In addition to the ongoing monitoring needed to achieve these objectives, there are often requirements to provide on demand insight into what is going on within the confines of the database system at any one moment in time. For some suggestions and examples for such situations, see: "Additional monitoring situations" on page 49.

Finally, for some other helpful thoughts and guidance on the topic of monitoring, with respect to workload management, see: "Other operational considerations" on page 51.

Monitoring system health

Monitoring your database system for signs of health is an important maintenance routine. The key indicators of a healthy system and guidelines on how to monitor them are provided here.

As described at the beginning of this best practices paper, the following are the key indicators that can be monitored to determine if the system is still in a healthy state:

- Run queue length is less than 10
- Overall CPU utilization is around 80-95%, with system CPU usage below 10-20%
  - The target value for system CPU usage varies by platform. For example, the target is less than 10% on Linux and less than 20% on AIX operating systems.
- Memory utilization is below 100% (that is, no paging)
- I/O waits are 10% or less
- System workload is evenly balanced across all members (that is, no skew or uneven resource demands)

The first three indicators are available through standard operating system interfaces such as the **vmstat** command on UNIX, AIX, and Linux systems, and **perfmon** on the Windows operating system. It is also possible to use the ENV_SYS_RESOURCES administrative view provided with the DB2 database manager to obtain some of the metrics that you want. However, as of DB2 for Linux, UNIX, and Windows Version 9.7 Fix Pack 4, it does not provide all of the metrics, hence requiring the use of additional interfaces as well. Use of the **vmstat** interface provides all of the required metrics in one invocation.

The monitoring requirement for these key indicators is simply to detect when the values are not within the target range for extended periods of time, without requiring a second-by-second view of their behavior. Ideally, a daemon or a repetitively scheduled script (such as a `cron` job) is established to acquire these values at a regular time interval and to store them for later analysis. This script can also proactively send alerts to the system operators to inform them when the current values significantly exceed the ideal values. For an example script, see: "Sample E1: Sample script for collecting vmstat data" on page 99.

This information is best collected with a time interval that is granular enough to give insight into small periods of less-than-ideal behavior and still large enough to allow sufficient history to be stored to cover periods of time where the data is not being actively analyzed (for example, over weekends and holidays). There is also a desire to coordinate this information with other information being gathered from the system, such as the workload management statistics. This coordination helps to pinpoint possible contributors to a problem, if it occurs. For this reason, use the same maximum interval as the one used for collecting data to the statistics event monitor, while the minimum interval can be much smaller, such as every minute.

The last piece of key information to watch for is any indication of skew in the demands on resources being made on the different members (also known as database partitions) of a partitioned database system. Noteworthy is any member on which there is a significantly larger or different demand for resources than on any of the other members. Excluding the administration member where applications connect to submit their work, all the other members in a partitioned database environment using the IBM Smart Analytics System methodology are typically designed to be identical in capacity and with equal portions of the data upon them; this design is done under the assumption and expectation that the resource demands upon them will also be identical. When the resource demands are not identical, this observation is often a sign that either there is a skew in the amount of data present, or that the application work is biased in the data it is accessing and the distribution of data might need to be revisited.

To search for signs of uneven resource demands on the members, the following sample report can be used to look within the workload management statistics data: "Sample F2: Database summary of work characteristics by service subclass" on page 111.

Monitoring activity behaviors

Once thresholds have been put in place to watch for unusual behavior patterns, there are two types of reports that become useful:

- Reports that help you maintain awareness and begin resolution of any activities that violate the thresholds
- Reports that help you watch for a change in the established norms for activity behavior over time so that the threshold definitions remain relevant

Awareness of threshold violations

Being aware of threshold violations is one of the key monitoring tasks to help control unusual activity behaviors through early detection. After violations have been detected, you can begin to resolve the underlying cause.

A useful type of report to help with this task is one based on the existence of threshold violations, as recorded by the threshold violations event monitor, and linking the violation data together with any relevant activity data from the activity event monitor. For an example SQL script of a summary report, see: "Sample F1: Summary report on threshold violations" on page 109. For information about how to receive email notifications whenever threshold violations occur, see: "How to generate email notifications for threshold violations" at http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.wlm.doc/doc/t0057200.html.

After you have identified that threshold violations have occurred, the next step is to attempt to resolve the underlying cause of the violations or to prevent, as early as possible, any processing capacity from being wasted by badly behaving statements. If activity data has been collected with the threshold violation, then this information can be used to evaluate what might have contributed to the behavior. The following information is key to such efforts:

Compilation environment
To see the environment used to compile the statement, use the COMPILATION_ENV table function on the COMP_ENV_DESV field in the ACTIVITYSTMT table of the activity event monitor.

Execution metrics
These metrics reflect when the activity ran and what its behavior was during execution.

Important: Activity details collected by a threshold violation are recorded on only the coordinator member, even if the violation occurred at a different member and if the ON ALL DATABASE PARTITIONS subclause is used as part of the COLLECT ACTIVITY DATA clause in the threshold definition. This means that any execution metrics captured in the event monitor by the threshold reflect only the perspective of the coordinator member. For a full database perspective, it is necessary to use the COLLECT ACTIVITY DATA clause on the connection (by using the WLM_SET_CONN_ENV procedure), workload, or appropriate service class before execution of the SQL statement in question.

- If using the DB2 Version 9.7 metrics controlled by the **mon_act_metrics** database configuration parameter, then the contents of the ACTIVITYMETRICS table can be analyzed (this table was introduced for the activity event monitor in DB2 Version 9.7 Fix Pack 4).
- If using the traditional system monitor switches and the STATEMENT monitor switch is ON, then the contents of the ACTIVITY table can be analyzed.

Access plan

The access plan is chosen by the SQL compiler to satisfy the SQL statement that is being executed.

If the SECTION key word is used in the COLLECT ACTIVITY DATA clause in the threshold definition, then the section is written out to the SECTION_ENV column of the ACTIVITYSTMT table of the event monitor when that threshold is violated. The access plan information in this column can be made available to the normal Explain table mechanisms through the use of the EXPLAIN_FROM_ACTIVITY procedure.

Note: If section actuals are being gathered, the section captured by a reactive threshold contains only information from the coordinator member perspective. For more information about section actuals, see: "Capturing and accessing section actuals" at http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.perf.doc/doc/c0056362.html.

The level of activity information available is determined by the setting of the COLLECT ACTIVITY DATA clause in the threshold definition with different tables in the event monitor being populated depending on the settings used in this clause. For additional information about what activity event monitor tables are affected by the different settings, see: "Logical data groups affected by COLLECT ACTIVITY DATA settings" at http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.mon.doc/doc/r0051558.html.

Watching for drift from the baseline norms

The behaviors of database activities can change gradually over time. Establishing one or more reports that report back on activity behaviors allow you to observe if there are any qualitative changes to the behaviors for activities.

Once detected, it then becomes an exercise to determine if the change is caused by some correctable situation within the application or database. Also a consideration is a fundamental change in the nature of the set of activities that is being watched, which might require an adjustment to the corresponding threshold definitions to decrease or increase their sensitivity.

The following is a list of key behavior attributes to be monitored for substantive changes over time:

- Concurrency rate
- CPU consumption
- I/O consumption
- Estimated cost
- Rows returned to application
- Lock wait
- Activity lifetime

- Unit of work duration

To have continuous monitoring of these characteristics, use of the statistics event monitor is recommended. Although the activity event monitor provides a large amount of information per statement execution and can be used to satisfy many other monitoring needs, it has a higher resource consumption in terms of processing and storage costs. The statistics event monitor collects more coarse, aggregate information, but at a much lighter overall cost. Both can be used together in many ways. For example, you can capture detailed activity data for a key low volume subset of work, and use the aggregate activity information to provide a global monitoring presence.

A reasonable interval for collecting statistical information is any interval in the range between 15 to 60 minutes. The actual interval period depends on the value, to your organization, of the increased granularity of insight versus the increased resource consumption costs (primarily in storage).

Two sample SQL scripts are provided for reports with a database perspective based on the information collected by the statistics event monitor (see: "Sample F2: Database summary of work characteristics by service subclass" on page 111 and "Sample F3: Database summary of work characteristics by workload" on page 114). Similar reports can be produced based on the contents of the activity event monitor by using standard SQL (use MIN and MAX aggregate functions on the fields of interest), but, unless activity data is captured at all members, such reports reflect only the coordinator member perspective of the work (resource consumption metrics are incomplete).

Two additional sample SQL scripts are provided for reports based on the information collected by the statistics event monitor and with a per member perspective (see: "Sample F4: Member summary of work characteristics by service subclass" on page 117 and "Sample F5: Member summary of work characteristics by workload" on page 122). These reports are often the next step in the process of deciding if a behavior change is localized to one member or systemic. The specific prerequisite monitor settings are identified for all of the reports.

Monitoring system behaviors

In addition to watching the key health indicators for the system and looking for abnormal activity behavior which might threaten system stability, monitoring system behaviors is another important part of the overall monitoring framework to put in place to help ensure the ongoing appropriateness of your workload management configuration.

The monitoring of system behaviors is focused on the following:

- Watching for changes in resource consumption or response time across service subclasses that might affect the targeted resource allocations
- Watching for changes in the estimated costs of incoming work that might affect the distribution of work across subclasses

The first objective is met by watching the rate of CPU and I/O resource consumption and the response time characteristics for each service subclass to see if there is a significant increase or decrease in either. When such changes are detected, then the same methodology used in the section entitled "Adjusting concurrency threshold values" on page 35 must be followed to adjust the configuration.

Note: It is important to confirm that any changes have stabilized (that is, have become constant) before adjusting the configuration.

The second objective is best satisfied by watching the distribution of estimated costs within each service subclass and the database as a whole to see if any of the scenarios arise that are discussed in the following topic and appendix: "Adjusting work class definitions" on page 34 and Appendix D, "Techniques for adjusting work class definitions," on page 81.

The expectation for these reports is not to spur improvised, repetitive changes to the workload management configuration in either of the aspects mentioned earlier. Rather, the expectation is primarily to detect any unexpected dramatic changes that might require action in the midst of gathering information to be used at the next regularly scheduled review of the workload management configuration.

Monitoring resource consumption

Two resources of most concern to database administrators are CPU and I/O. The monitoring goal is to know the actual consumption per service subclass over a period. The objective of this monitoring is to identify any service subclass whose resource consumption begins to change significantly because the result of this change is either a reduction in resources that are available to other service classes, or a surplus of resources that can be given for use elsewhere.

Of particular concern are those service subclasses that are not currently constrained by a concurrency threshold, because either a concurrency threshold does not exist for it, or the service subclass is not reaching its concurrency limit. Such behavioral changes need to be watched for their overall impact on the system. An increase in resource consumption can result in changed behavior in the other service classes; A decrease in resource consumption can lead to an opportunity to raise the concurrency limits of other service classes.

As in the previous section, the statistics event monitor presents a lightweight way to monitor the CPU and I/O resources being consumed by each service class through the **total_cpu_time** and **rows_read** metrics captured by the statistics event monitor in the DETAILS_XML column of the SCSTATS table[4]. For an example of such reports from database and member perspectives, see: "Sample F2: Database summary of work characteristics by service subclass" on page 111 and "Sample F4: Member summary of work characteristics by service subclass" on page 117.

Also of interest, although not directly applicable to the stage 2 workload management configuration, is the perspective of which workloads are consuming resources. This perspective can be achieved in a similar manner as previously described for service classes, except that you use the WLSTATS table. Such information is useful in determining if one or more workloads might need to be regulated in a manner similar to that described in the stage 3 "Regulating incoming work" scenario. For an example of such reports from database and member perspectives, see: "Sample F3: Database summary of work characteristics by workload" on page 114 and "Sample F5: Member summary of work characteristics by workload" on page 122.

4. Although the **rows_read** metrics provides a rough approximation of the I/O impact, for an alternative way to determine I/O impact that is based on pages read or written, see: Appendix G, "Alternative approaches to statistical data analysis," on page 127.

Monitoring estimated cost distribution

Changes in the distribution of estimated costs can happen over time due to a number of factors such as changes in the access plan, the amount of data affected, or the SQL statement itself. While changes in the resource consumption patterns are of a more immediate concern for the day-to-day maintenance of a stable database system, changes in the estimated costs indicate a change in demands being made on your system and potentially changes in population for the different service subclasses created as part of the best practices methodology. As such, they are an early indicator that perhaps a review of the workload management configuration is required to ensure continued stability.

Although changes in the high water mark for individual queries can be detected through use of the COST_ESTIMATE_TOP value in the SCSTAT table of the statistics event monitor, these changes do not give you information about shifts in the general population. To get this information, it is necessary to know about all of the costs that were encountered during the monitoring interval. Information about the costs can be obtained in the following two ways:

- Using the activity event monitor on each service subclass, in the same manner as described in the earlier stages of this best practices document, to collect information about each SQL statement executed.
- If DML SQL statements are not issued from within a CALL statement (nested SQL statements do not exist in your environment), the use of the COLLECT AGGREGATE ACTIVITY DATA clause with the EXTENDED value on each service subclass results in a histogram of the encountered estimated costs.

For more information regarding how to evaluate the distribution of estimated costs, see: "Gathering detailed monitoring information for adjusting work class definitions" on page 32.

Additional monitoring situations

Ad hoc monitoring

Aside from any regular background monitoring, there is often a need, during daily operations, for active, on-demand insight into what is occurring within the database system at the current time to resolve or better understand an issue. A number of features are provided within the DB2 database manager to provide such insight. This section highlights a few of these features that can be used to answer common questions or deal with common scenarios.

To determine what is currently running on the system, the MON_CURRENT_SQL and MON_CURRENT_UOW views are good places to start your investigation. These views, as listed in Table 8, provide the salient identification and resource consumption details about all the work running on the system at the time of your investigation. From this information, more detailed or targeted queries can be framed to drive out the critical details needed to resolve the situation at hand.

Table 8. Sources of information for different areas of interest

Area of interest	Obtain current information	Obtain historical information
Requests	WLM_GET_SERVICE_CLASS_AGENTS_V97 table function	Statement event monitor

Table 8. Sources of information for different areas of interest (continued)

Area of interest	Obtain current information	Obtain historical information
Activities (SQL)	One or more of the following: • MON_CURRENT_SQL view • MON_GET_ACTIVITY_DETAIL table function • MON_GET_PKG_CACHE_STMT table function • WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES_V97 table function	Activity event monitor
Transactions	One or more of the following: • MON_CURRENT_UOW view • MON_GET_UNIT_OF_WORK table function	Unit of work event monitor
Connections	One or more of the following: • MON_CONNECTION_SUMMARY view • MON_GET_CONNECTION table function • WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 table function	Connection event monitor
Workloads	One or more of the following: • MON_WORKLOAD_SUMMARY view • MON_GET_WORKLOAD table function • WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES_V97 table function • WLM_GET_WORKLOAD_STATS_V97 table function	Statistics event monitor
Service subclasses	One or more of the following: • MON_SERVICE_SUBCLASS_SUMMARY view • MON_GET_SERVICE_SUBCLASS table function • WLM_GET_SERVICE_CLASS_AGENTS_V97 table function • WLM_GET_SERVICE_SUBCLASS_STATS_v97 table function	Statistics event monitor

One or more of the other table functions, contained in Table 8 on page 49, can also be used to build customized queries or scripts to dive down into many of the details of what is currently occurring within the database system. When deep-diving for details, the following are the key identification fields (column names) to be used to join between the output of the various table functions:

- APPLICATION_HANDLE
- UOW_ID (where applicable; must be used with APPLICATION_HANDLE)
- ACTIVITY_ID (where applicable; must be used with UOW_ID and APPLICATION_HANDLE)

Given the use of concurrency thresholds in the workload management best practices, you might also be interested in the current state of activities that might be queued. For an example that explains how to get information about queued activities, see: “Example: Determining which activities are queued by a WLM threshold and their queue order” at http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.wlm.doc/doc/c0056919.html.

As noted earlier in this document, the activity event monitor is the best way to collect detailed historical information about the execution of SQL statements. In addition to using the COLLECT ACTIVITY DATA clause as an explicit way to indicate in the workload management configuration that you want to gather this information, you can also get this detailed information through the use of the following stored procedures:

- WLM_CAPTURE_ACTIVITY_IN_PROGRESS stored procedure to explicitly identify the activity of interest
- WLM_SET_CONN_ENV stored procedure to modify the collection settings of any connection, without the need for a separate workload definition, to cause the data to be collected

Other operational considerations

This section points out a few other key considerations that must be taken into account when implementing an ongoing monitoring regime.

Space management with the statistics event monitor

One aspect of the statistics event monitor that could become an issue on systems with a large number of workload management entities that are collecting aggregate activity data is the `histogrambin` table. This table can experience a large amount of growth even with just the monitoring configuration suggestions provided with this best practices document.

Although these histograms are collected at every member for each workload management entity which has specified that aggregate activity data is to be collected, for the histograms of general interest (that is, CoordActQueueTime, CoordActExecTime, CoordActLifetime, and CoordActEstCost), we care about only the data gathered at the coordinator member where the connections are established by the applications.

The following is a list of some space management techniques that can be used to manage the amount of space consumed by the histogram data:

- If available, apply compression to the `histogrambin` table. In fact, all of the tables in the statistics event monitor respond well to compression.
- Schedule a script to regularly prune the histogram data from non-coordinator members. This histogram pruning is in addition to any existing process to prune the event monitor data as a whole and can be done on a more frequent basis to accommodate your unique storage restrictions.
- If you are copying the data to a different location, copy only the `histogrambin` information from the coordinator members. If you are not sure which are the coordinator members, then merge the data across all of the partitions using the following example SQL statement and copy the result:

```
SELECT PARTITION_KEY, BIN_ID, BOTTOM, HISTOGRAM_TYPE, SUM(NUMBER_IN_BIN),
       SERVICE_CLASS_ID, STATISTICS_TIMESTAMP, TOP,
       WORK_ACTION_SET_ID, WORK_CLASS_ID, WORKLOAD_ID
FROM HISTOGRAMBIN_DB2STATISTICS
GROUP BY PARTITION_KEY, BIN_ID, BOTTOM, HISTOGRAM_TYPE,
         SERVICE_CLASS_ID, STATISTICS_TIMESTAMP, TOP,
          WORK_ACTION_SET_ID, WORK_CLASS_ID, WORKLOAD_ID;
```

- If you know which members are the coordinator members, then define a table space that only exists on these members and create a separate event monitor that only captures the histogram table in that table space. Then, merge the output of the two event monitors as needed when accessing (or copying) the data.

For an example of this last point, assuming the existence of a TS_COORD_ONLY table space that exists only on coordinator members, the following statements are modified versions of the sample DDL statement provided elsewhere in this document. These versions create two event monitors; one event monitor gathering all of the information, except histogram data, on all members, and one gathering only histogram data existing on coordinator members:

```
CREATE EVENT MONITOR DB2STATISTICS1
FOR STATISTICS
WRITE TO TABLE
  SCSTATS (TABLE SCSTATS_DB2STATISTICS IN TS_MONITORING),
  WCSTATS (TABLE WCSTATS_DB2STATISTICS IN TS_MONITORING),
  WLSTATS (TABLE WLSTATS_DB2STATISTICS IN TS_MONITORING),
  QSTATS (TABLE QSTATS_DB2STATISTICS IN TS_MONITORING),
CONTROL (TABLE CONTROL_DB2STATISTICS IN TS_MONITORING);

CREATE EVENT MONITOR DB2STATISTICS2
FOR STATISTICS
WRITE TO TABLE
  HISTOGRAMBIN (TABLE HISTOGRAMBIN_DB2STATISTICS IN TS_COORD_ONLY),
CONTROL (TABLE CONTROL_DB2STATISTICS IN TS_COORD_ONLY);
```

Event monitor maintenance

When requested, the DB2 database manager generates and outputs data to an event monitor, but it does not manage that data once it has been written to disk. It is the responsibility of the event monitor owner to manage the data in terms of a backup or archival policy, if any, and how long the data is kept.

There are a number of ways to manage the data, but there are some common guidelines that apply to all approaches. The most important thing to remember is that while an event monitor is active, it holds an intention exclusive (IX) table lock on any tables to which it is writing information to prevent those tables from being dropped while it is using them. As a result, any action that potentially collides with that lock can have undesirable consequences, such as negatively impacting system performance or interfering with your monitoring capabilities. Such actions include the following:

- Backup
- Mass deletion by using an unbounded DELETE statement (due to lock escalation)
- Truncation of the table

The easiest way to deal with this potential collision is to set up your event monitor so that it *toggles* between two event monitors with only one being active at a time. That is, after the second event monitor is activated, the first one is deactivated and then the actions that you want are performed upon the data of the first event monitor at that time. While there is some chance for duplicate events being recorded during the brief moment when both event monitors were active at the same time, these duplicates can be removed to keep the data clean.

If this approach is not what you want, then a single event monitor can be left active with subsets of data being extracted to a secondary site where it can be analyzed or backed up as you want and then deleted. This data collection can be done by using SELECT and DELETE statements with predicates that restrict the data being considered to a limited subset of data (to avoid lock escalation) from the past (to avoid collision with any new rows being inserted). When deleting, consider setting a lock timeout before issuing the DELETE statement (for example, SET CURRENT LOCK TIMEOUT 60) to automatically resolve any collision between the delete statement and the normal event monitor operations. If

increasing the lock timeout period does not resolve the problem, try deleting smaller subsets of the data such as the records for shorter time periods. This approach requires fewer locks, which reduces the chance of lock escalation (and collision) occurring.

Analyzing statistical data by running sample SQL scripts

Although the example SQL scripts, that are provided in this document, are designed to run directly against the statistics event monitor tables, the scripts are more a matter of convenience, rather than a recommended statistical analysis method, for the following reasons:

- Performance of the queries can be poor, when a large number of rows exist, due to redundant calculations and the lack of indexes
- There is the potential for lock conflict when new records are being inserted by an active statistics event monitor

For this reason, it is recommended that statistics event monitor data analysis occur on data that is not actively being accessed by the event monitor. It is also recommended that the data has been modified, to better support the analysis phase, through the introduction of indexes and precalculation of any common information needed by that analysis. For some examples of SQL scripts that might be of use in such endeavours, see: Appendix G, “Alternative approaches to statistical data analysis,” on page 127.

Advanced configurations: Stage 3 scenarios

As defined previously, a stage 3 workload management configuration is simply any configuration that exceeds or differs significantly from a stage 2 configuration. These significant extensions are required when dealing with unique requirements that go beyond basic system stability. It is during the transition to a stage 3 configuration that the unique business challenges and corporate perspectives or strategies of each individual DB2 implementation begin to heavily influence the workload management configuration, making it difficult to provide a common stage 3 configuration template for use by all.

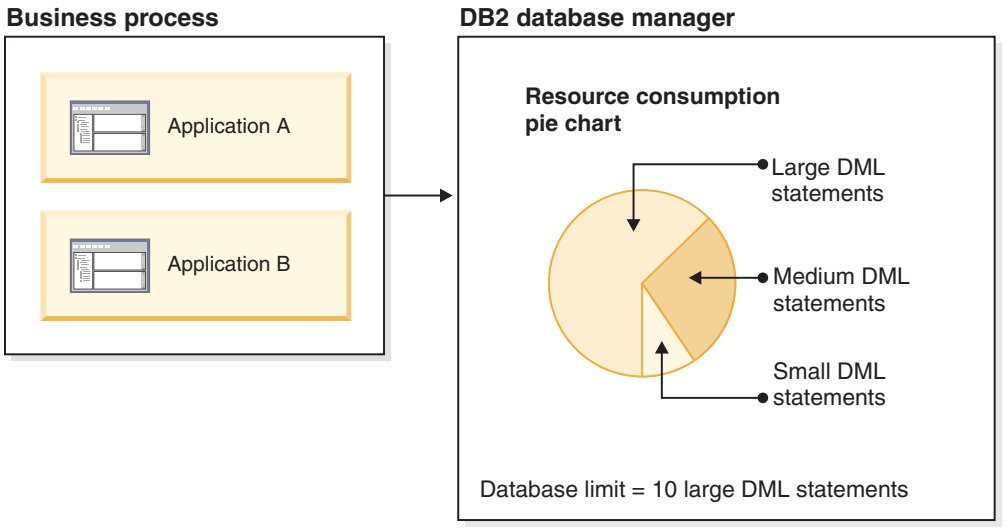
Instead, this document provides a brief description of the most common stage 3 workload management scenarios encountered, along with some guidance on the general best practices implemented when a similar scenario is encountered.

Scenario: Regulating incoming work

After reaching a stable stage 2 workload management configuration and the overall mix of work allowed on the system at any one time has been determined, it is now possible to decide how much of that mix, and the system resources assigned to each work type, can be consumed by any specific application or business area. That is, you can now begin to look at controlling the demands put on the system by individual sources of work.

This control plan usually involves a business discussion about the budget allocation of database resources to be consumed by the individual business processes or applications. This discussion has the outcome of improving the performance of some of the business processes or applications based on their relative importance to the business as a whole. In other words, you are deciding the *fair share* resource allocations between the competing processes and applications.

The natural location within DB2 workload management for controlling how much work any individual source can submit to the database is the DB2 workload as shown in Figure 10 on page 56. By identifying the source of work with a specific DB2 workload definition, not only can you monitor the performance and demands of that work, but you can also control the quantity and type of work the workload definition can submit.



Note: Application B is restricted to 2 large DML queries at a time

Figure 10. Allocating portions of work across different business areas

For example, by applying a work action set to a workload, we can impose concurrency restrictions on the type of work allowed to be submitted, meanwhile keeping the overall system perspective simple. Figure 11 shows an example of how allocation of resource consumption across business areas can be introduced without changing the service class definitions introduced previously.

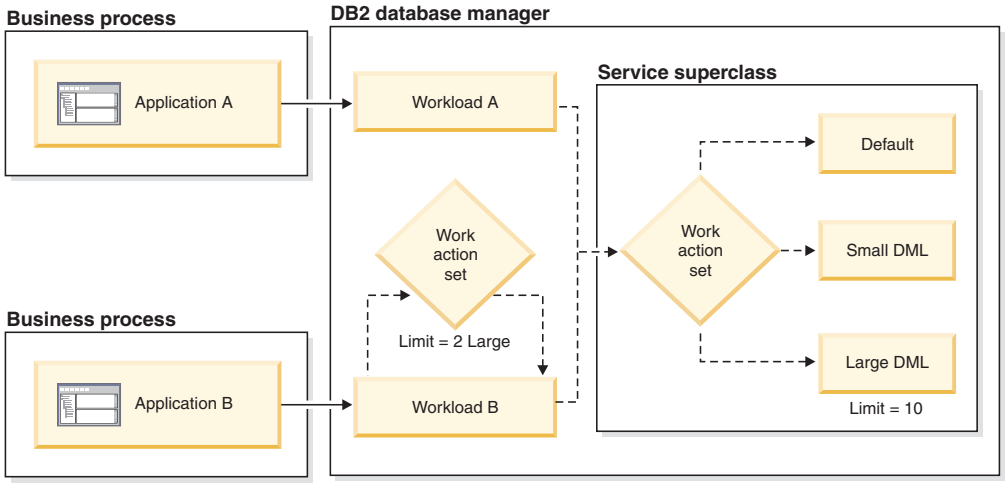


Figure 11. Example of allocating consumption between business areas

Scenario: Protected work

This scenario features a specific set of work that is favored and is to be protected from the resource demands of other work. In some cases, this same protected work is also given first preference whenever excess resources are available.

This type of scenario often arises whenever a mission-critical, performance-sensitive application coexists with other less critical applications on the same database system. In such cases, the priority is to ensure the responsiveness of the

critical application, regardless of the type of work it is submitting, while making the resource needs of other work a secondary consideration.

When you have such a situation, the recommended course of action is to start by following the process to reach a stable stage 2 configuration and to create a unique workload definition or definitions to allow individual control and monitoring of the critical application. The rationale behind this guidance is that the impact of the best practices template on the critical application depends entirely on the type of work being submitted. If the work submitted falls into those service subclasses that do not have concurrency limits placed upon them, then the performance attributes of the critical application might be satisfactory without any additional modifications to the existing configuration.

However, if the critical application is not making its performance objectives within the standard best practices template configuration, then it is necessary to pull that work out of the service superclass of the best practices template configuration, place the work of the critical application into its own service superclass and isolate it from the others, as depicted in Figure 12.

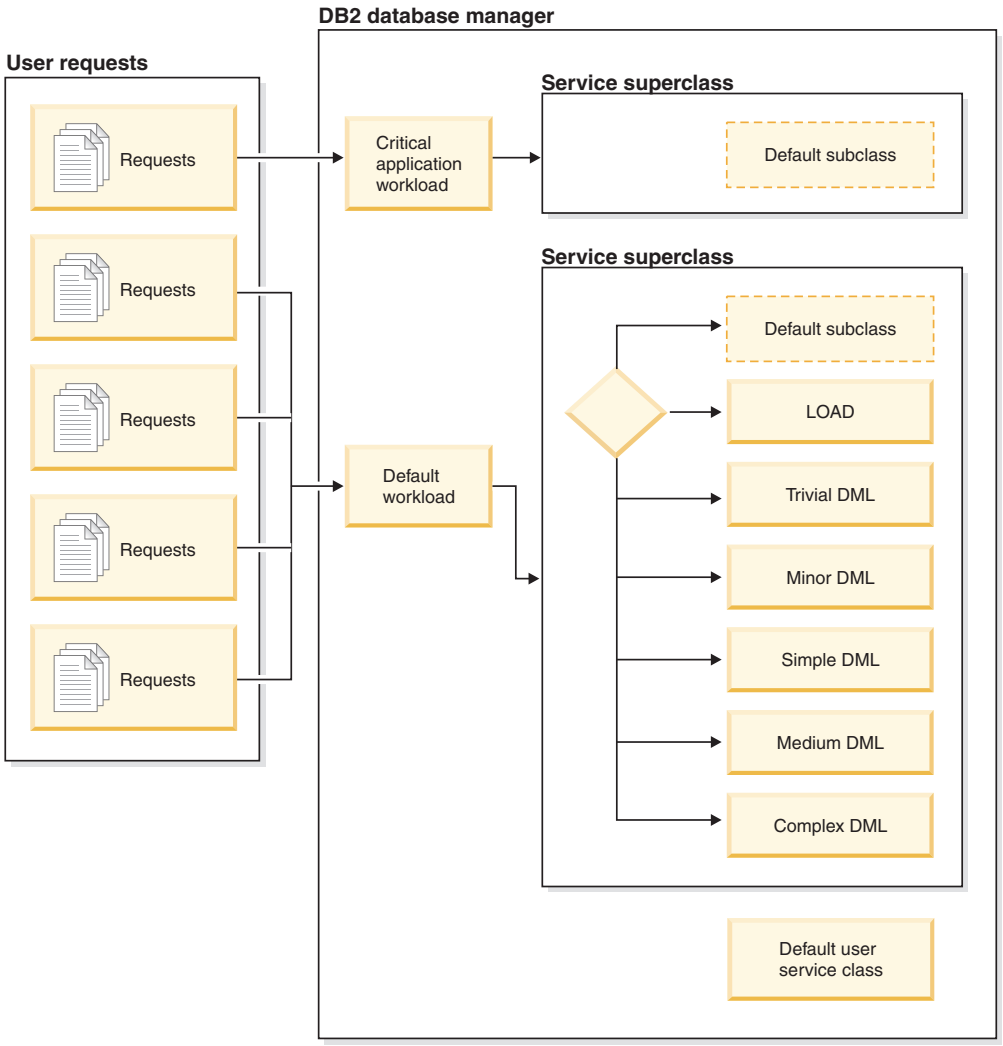


Figure 12. Critical application workload isolated in its own service superclass

In this configuration, it is possible on AIX platforms to further enforce the isolation of CPU resources between the different service superclasses by using AIX WLM hard limits. Using these controls, you can strictly control the amount of CPU resources consumed by one or both of the two service superclasses shown in Figure 12 on page 57. By creating and associating an AIX service superclass with one or both of the DB2 service superclasses, it is possible to favor one superclass over the other, or help to ensure that both get consistent access to CPU resources.

In Figure 13 and Figure 14, we have a simplified representation of the basic scenario. We have the service class containing the favoured work to be protected and one or more of the other service classes. Figure 13 shows that if other service classes are limited by an AIX WLM hard limit, then the critical or favored work is guaranteed a minimum amount of CPU with the ability to access the rest of the system.

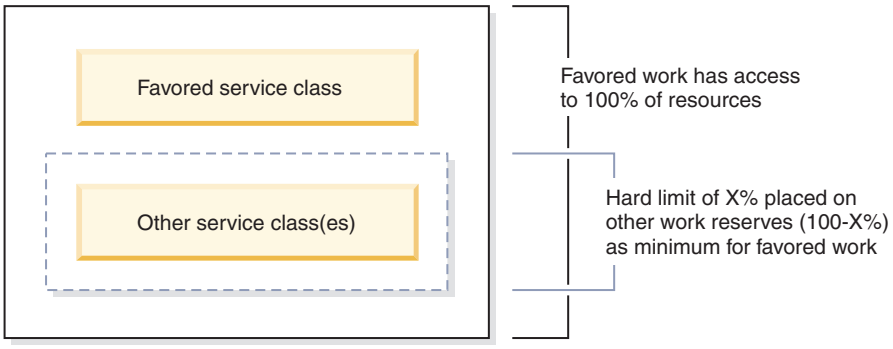


Figure 13. Critical or favored work can access all system resources

In Figure 14, we have the case where both the favored service class and the other ones are limited by AIX WLM hard limits, then both are guaranteed a consistent CPU resource allocation.

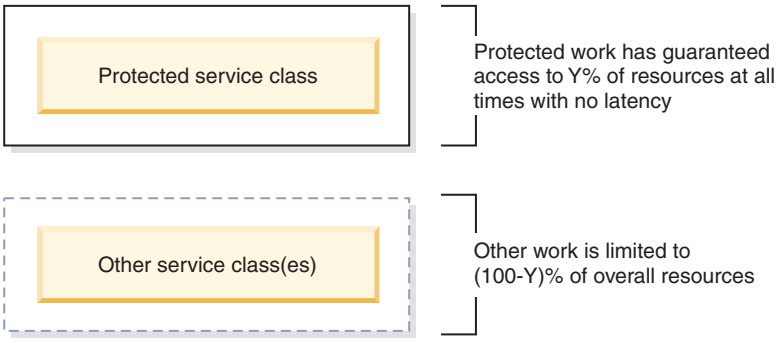


Figure 14. Consistent throughput scenario

For more information about using AIX WLM hard limits, see the best practices supplement: "Integrating DB2 Workload Management with Operating System Workload Management" in the `OS_WLM.pdf` file that is included in the best practices `.zip` package. The use of soft limits (also known as shares) on AIX or Linux is typically not useful in these scenarios because such controls are only effective when the overall system CPU utilization is quite high. Otherwise, soft limits are not active, whereas hard limits are active at all times, regardless of the CPU utilization rate.

Scenario: Production shifts

Another common scenario that goes beyond the standard stage 2 configuration is one where a business encounters one or more periods of significantly different demands being placed on their database system. The simplest example of such a situation is the transformation from a daytime *online* environment to an overnight *batch* environment. Other examples can include changes in system usage by different parts of the enterprise due to the demands of monthly or quarterly business cycles, such as quarterly reporting requirements. Effectively, any known and predictable change in the mix of work, priorities, or performance demands made on the database system could require a change in the current workload management configuration to recognize the new conditions and expectations.

The term *production shifts* was coined to identify such scenarios, with each unique set of demands being considered as a different *shift*.

In such situations, the following challenges are encountered:

- Determining the best workload management configuration for each shift
- Switching the workload management configuration for the system at the start of each shift

If all of the target environments consist of work that maps well to the underlying assumptions of the best practices (the workload consists primarily of DML statements whose costs correspond with the resource impacts), then the initial recommendation for addressing the first challenge is to treat each shift as a unique exercise and to follow the best practices methodology to achieve a stable stage 2 configuration for each set of conditions. The result gives you the two different configurations that you want to have in place on your system at different times.

After you have defined the set of configurations that is to be used during the different shifts and the time frames in which each configuration is to be running on the system, then the final challenge is how to best transition between the different configurations. Best practices in this area recommend the use of the extensive ALTER capabilities available with DB2 workload management to minimize disruptions to the system. Using the various ALTER statements, it is possible to make the following changes online with minimal impact to the system:

- Altering a workload definition to point to a different service class
- Altering a service class definition
- Altering limits for an activity or concurrency threshold
- Altering a threshold, workload, or service class to enable or disable them

If you are using AIX WLM or Linux WLM capabilities, you can alter the definitions of these entities outside of the DB2 database manager, but in a partitioned database environment, an alteration requires changes on each and every member involved in the database. Coordinating such changes across many members can be difficult. An alternative, and vastly simpler, approach is to predefine a set of unique OS WLM definitions for each shift, each with its own external correlation token or tokens. When the change is needed, alter, with the ALTER SERVICE CLASS statement, the involved DB2 service classes to use a different external correlation token.

The workload management tooling provided in the Optim Performance Manager (Optim PM) can be used to create more complex scripts for altering configurations. You can create more complex scripts by defining the first configuration to Optim

PM and then altering that configuration to the other settings. Optim PM produces the necessary script to transition between the two configurations.

Scenario: Tiered service offerings

In some environments, business strategy has led to the desire to offer database users different levels of service based on either their importance to the business or ability to pay. Typically, the different levels of service are distinguished by different levels of resources made available to the level of service.

For example, a premium service level is offered with capped access to 60% of the system resources, a standard service level with capped access to 30%, and a substandard service level of capped access to only 10%. The result is that applications in different service levels experience different levels of performance.

The best practices recommendation is to duplicate the best practices template for each tier of service, as shown in Figure 15, and follow the general methodology outlined in this document.

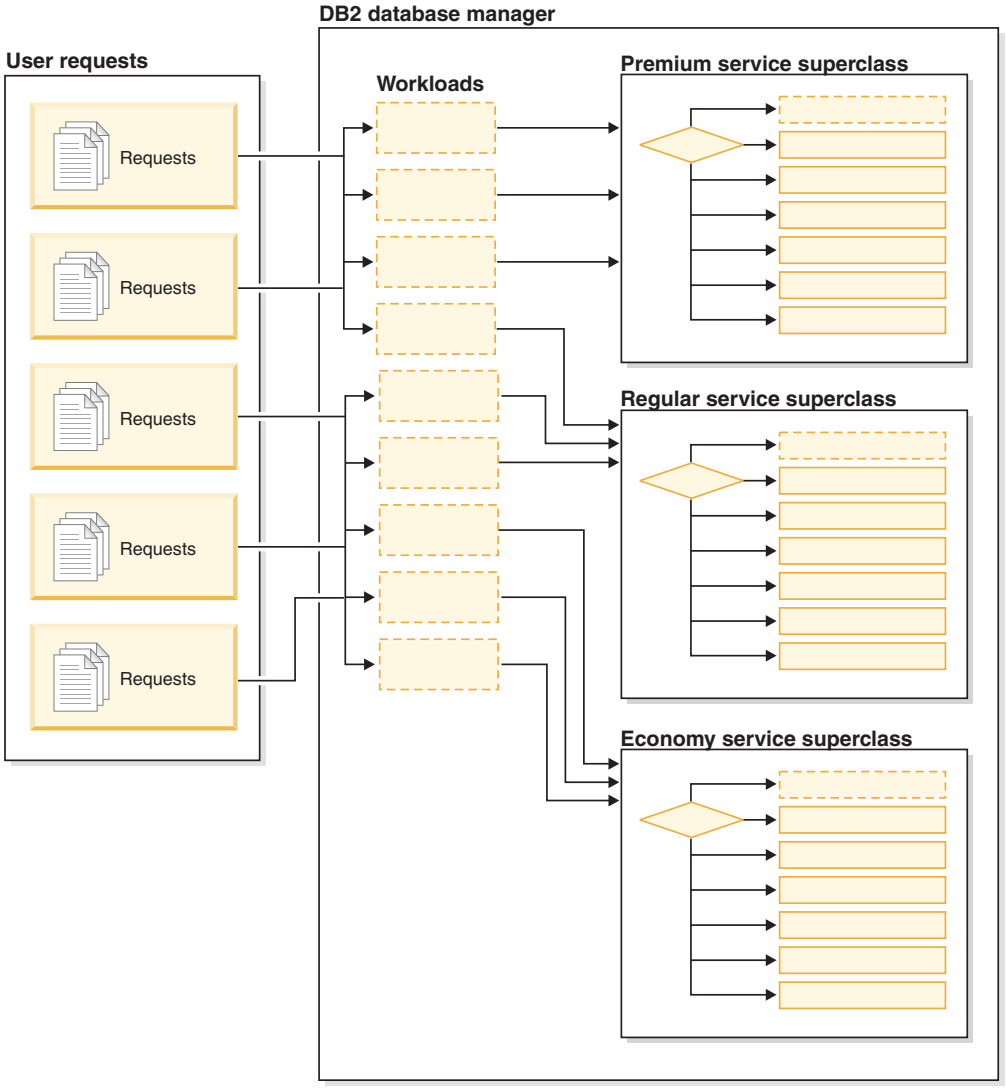


Figure 15. Tiered service offerings

The major difference is that the overall system capacity, calculated as a starting point for concurrency, needs to be suballocated across the different service classes before the concurrency thresholds can be set on the individual service subclasses within each superclass. For example, if the system is calculated to support a concurrency of 180, 100 of that could be assigned to the Premium service superclass, 60 to the Regular service superclass, and the remaining 20 to the Economy service superclass. Using these initial system capacities for each service superclass, the standard methodology can be used, although different budget approaches can be taken within each service class to allocate the assigned capacity across the service subclasses.

The one caveat to this approach is that while the overall concurrency numbers of the different work types cannot exceed the overall system capacity, the demands on the system might come from different service superclasses at any one time. The result of subdividing the overall capacity between the different service superclasses is a configuration that is less flexible to a fluctuating workload. In such cases, it might be desirable to have slightly higher concurrency limits (or even none) on one or more of the service subclasses in each service superclass, and then apply a database-level concurrency control on the same work to reflect the system capacity.

For this approach to work, it is necessary that all of the participating service subclasses contain work of the same estimated cost range. For example, if the overall configuration is designed to accommodate six complex queries (queries that have an estimated cost that falls into the complex range), then a work action set is defined at the database level to impose a concurrency threshold of 6 on queries with an estimated cost in the complex range. The concurrency thresholds on the individual service subclasses that represent that same range in each service superclass can be set to values different from 6, if you want[5]. Such an arrangement controls the overall use of the system by complex queries, and the participation of the individual corresponding subclasses can be curtailed as you want.

One other consideration that needs to be thought about is the mapping of workloads to service superclasses in a tiered service system. In the simplest of cases, each application or business unit maps to only one service level, resulting in single workloads being used to represent both the business entity and the mapping to the service level. In more complicated cases, different aspects of each business application or unit map to different service levels. This environment requires a workload to be defined for each relationship between an outside entity and a service level, with connections (or transactions) being mapped to the appropriate workload for the service level. That is, if an ACCOUNTS application had two types of database interactions, one with the Premium level and one with the Economy level, then two workloads need to be created for the ACCOUNTS application, one pointing to the Premium service superclass and one pointing to the Economy service superclass. In addition, appropriate changes need to be made within the ACCOUNTS application context to ensure that transactions or connections are mapped to the appropriate workload. Also, summary reports, of database activity for the ACCOUNTS application, need to be aware of the existence of multiple workloads representing that application within the database and integrate the different data into one cohesive report.

5. Note that the concurrency threshold imposed on the complex work by the database work action set is evaluated first before the relevant service subclass concurrency threshold is evaluated. For this reason, it is important to be careful when having service class concurrency thresholds with limits lower than the database one since this might result in queries that are permitted to execute by the database threshold being made to wait by the service class threshold. The best practice is to have the database threshold be more restrictive than the service class one.

Scenario: Non-CPU contention

Although this best practices document has been focused on the control and allocation of CPU resources between the different types of work being processed by a database system, there are cases where the contention between work is caused by resources other than the CPU. This *non-CPU contention* can be caused by contention between two or more applications for system resources such as I/O, memory (for example, sort memory), or even storage (for example, system temporary table space). That is, two or more workloads are colliding over access to a non-CPU resource and this collision is affecting the efficiency of the system.

If you are unable to address the contention for non-CPU resources in any other way, it is possible to use a workload management configuration to help in such a situation. By taking advantage of the *gatekeeper* aspect of the concurrency thresholds, it is possible to indirectly control the demands made on resources, other than the CPU, by adjusting the concurrency limits. Work that is not executing, because it is queued in the concurrency threshold, will not be making demands on the contentious resource. For example, low priority work is consuming too much sort memory and affecting the performance of higher priority work (perhaps by causing its sorts to spill to disk). By lowering the overall concurrency of the low priority work with a concurrency threshold, you can reduce its overall demands on sort memory because sort memory is only allocated for the low priority work after its execution has started.

The following is a rough outline of the steps required to address contention for non-CPU resources by using concurrency controls in DB2 workload management:

1. Identify what resource is in contention and obtain monitoring information for it from the different workloads and service classes in your configuration.
2. Identify what type of work is causing the contention. If it is the execution of SQL statements or the LOAD utility, continue; otherwise, stop because you cannot resolve this issue by using concurrency thresholds that only act on recognized activities. For more information about these recognized activities, see: "Prerequisite concepts and terminology" on page 5.
3. If the problem is systemic (that is, involving all workloads), then adjust the configuration, as a whole, as was done for CPU. Review the section about adjusting concurrency threshold values by using the consumption rate for the resource in question rather than just the CPU resource. See: "Adjusting concurrency threshold values" on page 35.
4. If the problem is not systemic, then identify which specific connections or transactions are competing for the resource and determine their business priority relative to each other.
5. Create a separate DB2 workload for each competing (or over-consuming) connection or transaction to allow that work to be monitored and controlled differently than all the others.
6. Identify if a specific class of work is responsible for the contention (for example, large queries consuming lots of sort space).
7. If the problem is caused by a specific set of work, perform the following actions on the lower priority DB2 workload definition:
   a. Create a DB2 work class set with one work class definition identifying the specific problematic subset of activities. For example, if the problem set is large queries, define a work class for large queries.
   b. Create a DB2 work action set on the lower priority DB2 workload definition that implements a concurrency threshold on the problematic work class

defined in the previous step. This action limits the concurrency level of all activities that match this work class definition coming from connections that map to this workload.

8. If the problem is not limited to a specific class of work, perform the following actions on the lower priority DB2 workload definition:
   a. Create a DB2 work class set with one work class definition of type ALL.
   b. Create a DB2 work action set on the lower priority DB2 workload definition that implements a concurrency threshold on that one work class defined in the previous step. This action limits the concurrency level of all recognized activities coming from connections that map to this workload, regardless of their different individual work types.

Conclusion

The workload management best practices presented in this paper are motivated by the goal of achieving more consistent and predictable performance for a database system deployed in a data warehouse environment. The objective of this document was to provide a standardized methodology that can be used by our customers to achieve that goal.

To this end, this document introduced an approach to achieving a customized stage 2 workload management configuration for a DB2 for Linux, UNIX, and Windows Version 9.7 database. This approach consists of the application of a *template* stage 1 workload management configuration to the target database, accompanied by a description of the steps needed to customize that template to better fit the actual work being executed on the target database.

After being established, this stage 2 workload management configuration can help reduce the volatility in performance that is experienced when the demands on the database exceed the available capacity by controlling the specific mix of work allowed to execute at any one time. To maintain this configuration, this document outlines a suggested monitoring regime to be put in place after the stage 2 configuration has been achieved. This monitoring regime aids in the future detection of change and in any subsequent increment tuning that might be required to compensate for that change.

For those customers with more advanced workload management requirements for their database systems, we have briefly outlined a set of common scenarios that have been encountered where more advanced, stage 3 workload management configurations have been required to address the requirements. Although the intent of this paper was not to provide specific guidance in these cases, it is hoped that the examples are of some aid to any customer contemplating such changes.

In the end, like many other parts of any IT environment, a database workload management configuration will ultimately reflect the unique business objectives and priorities of each enterprise. A standard approach to workload management, such as laid out in this document, can be extremely helpful to those customers who might be inexperienced or new to the topic, particularly in getting started and dealing with basic issues. Having a solid foundation, from which to begin to address more advanced requirements, reduces the set of issues and concerns that must be dealt with when grappling with the topic of workload management. If you follow the field-tested best practices outlined here for DB2 workload management, you will be off to a successful start to implementing workload management in your database environment.

Further reading

Additional information about DB2 workload management and related topics is provided here and can be used for both introductory and supplemental purposes with respect to the contents of this document.

Best practices for DB2 for Linux, UNIX, and Windows
http://www.ibm.com/developerworks/data/bestpractices/db2luw/

DB2 9.7 documentation
http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

DB2 9.7: Using Workload Manager features (Tutorial)
http://www.ibm.com/developerworks/data/tutorials/dm-0908db2workload/index.html

DB2 Workload Management Histograms,

Part 1: A gentle introduction to histograms
http://download.boulder.ibm.com/ibmdl/pub/software/dw/dm/db2/dm-0810mcdonald/dm-0810mcdonald-pdf.pdf

Part 2: Understanding the six histograms of DB2 workload management
http://download.boulder.ibm.com/ibmdl/pub/software/dw/dm/db2/dm-0810mcdonald2/dm-0810mcdonald2-pdf.pdf

Part 3: Visualizing and deriving statistics from DB2 histograms using SQL
http://download.boulder.ibm.com/ibmdl/pub/software/dw/dm/db2/dm-0810mcdonald3/dm-0810mcdonald3-pdf.pdf

White paper: Workload Management with MicroStrategy Software and IBM DB2 9.5
http://public.dhe.ibm.com/software/data/sw-library/db2/papers/wlm_msi_db29.5.pdf

SAP and DB2 for LUW: Exploitation of DB2's Workload Management in an SAP Environment
https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/d046f3f5-13c5-2b10-179d-80b6ae7b9657

IBM Smart Analytics System
http://www-01.ibm.com/software/data/infosphere/smart-analytics-system/

Contributors

Significant contributions were made to the theme and content of this workload management best practices paper by the following contributors:

John Bell

IBM Distinguished Engineer

Data Warehouse Architect

Kevin Beck

Software Developer

Data Warehouse Edition Development - Workload Management

Acknowledgements

We wish to acknowledge the valuable contributions of the following people who helped to improve the quality of this workload management best practices paper:

Maksym Petrenko

Malcom Zung

Challen Pride-Thorne

Sandra Seenauth

Garrett Fitzsimons

Paul McInerney

Serge Boivin

Eric Koeck

Karen McCulloch

Patrice Turpin

Scott Walkty

Steve Rees

Katherine Kurtz

Arun Lakhana

Eric Sirois

Jackie Chong

Appendix A. DB2 workload management and monitoring highlights for DB2 for Linux, UNIX, and Windows Version 9.7

The release of DB2 for Linux, UNIX, and Windows Version 9.7 continued the major investments in workload management and monitoring infrastructure in the DB2 engine with both the DB2 Version 9.7 and subsequent fix pack releases. Significant enhancements were made to workloads, thresholds, and service classes. In addition, the DB2 Version 9.7 release officially deprecated the Query Patroller product and the DB2 Governor and they will be discontinued in a future release.

As a high-level summary, the DB2 Version 9.7 release contained the following:

- The option to integrate DB2 service classes with Linux WLM
- Support for defining DB2 thresholds directly on DB2 workloads
- Support for the collection of aggregate activity data for DB2 workloads
- Changes to the behavior of the CONCURRENTDBCOORDACTIVITIES threshold
- The introduction of new DB2 thresholds for CPU and I/O consumption by SQL statements (for example, CPUTIME, SQLROWSREAD)
- The introduction of a new DB2 threshold for the aggregate consumption of system temporary table space by all Data Manipulation Language (DML) statements in a specific service class (AGGSQLTEMPSPACE)
- Support for a new REMAP action within (specific) DB2 thresholds to move activities between different subclasses (CPUTIMEINSC, SQLROWSREADINSC)
- The introduction of a new service class attribute to influence buffer pool I/O behaviors (BUFFERPOOL PRIORITY)
- Support for the collection of the new in-memory metrics, including wait time metrics, at the activity, workload, and service class levels, with corresponding MON_* table functions introduced to provide access
- Numerous enhancements to the DB2 workload management table functions and event monitors

In addition to the preceding enhancements, the DB2 for Linux, UNIX, and Windows Version 9.7 Fix Pack 1 release added the following:

- Support for the definition of work action sets directly on DB2 workloads with the ability to apply thresholds or collect aggregate activity data with a work action
- Changes to synchronize the automated collection of workload management statistics to a standard start time
- The introduction of a tool to help in the migration from Query Patroller to DB2 workload management
- The introduction of a new DB2 threshold to limit the length of time that a unit of work can remain active (UOWTOTALTIME)
- Support for the collection of the new component time metrics at the activity, workload, and service class levels
- Support for the collection of sections, executable form of SQL statements, as part of the activity event monitor for use with the new Explain from section capabilities which include capture of section *actuals*

Subsequent fix packs have added the following features:

- WLM_SET_CONN_ENV and WLM_GET_CONN_ENV procedures to allow the control of workload management monitoring information at the individual connection level (DB2 Version 9.7 Fix Pack 2)
- Add the event_activitymetrics logical data group to the activity event monitor to provide easy SQL access to the new DB2 Version 9.7 metrics in a separate table (DB2 Version 9.7 Fix Pack 4)
- An enhancement to the WLM_COLLECT_STATS procedure to provide a new input parameter to have it wait until all event monitor records have been flushed to disk before it returns, as well as a new output parameter to return a timestamp for statistics that were recently collected (DB2 Version 9.7 Fix Pack 4)

Appendix B. Creating prerequisite event monitors

Sample scripts are provided in this appendix to create prerequisite event monitors after reaching the stage 1 best practices template configuration.

In order to follow the methodology laid out in this document, it is necessary to create the following event monitors:

- Activity
- Statistics
- Threshold violations

These event monitors are required to be created in a table space that is present on all members, including the administration partition. Typically, this table space would include any regular table space created in the IBMDEFAULTGROUP database partition group. On IBM Smart Analytics System configurations, the event monitors can be created in the existing TS_MONITORING table space which is created across all members.

The following is an IBM Smart Analytics System example of the DDL that can be used (for an editable version of this SQL script, you can use the `sample_create_evmon.sql` file included in the best practices `.zip` file):

```
--
-- Script to create event monitors used by the DB2 Workload Management
-- Best Practices for Data Warehouses
--
-- NOTES:
--       1) Replace TS_MONITORING tablespace name if not appropriate for your
--          environment with the name of a tablespace that is accessible by all
--          database members
--
--       2) This DDL is compatible with DB2 9.7.4 for Linux, Unix, and Windows.
--           - If used with a version below this, remove the ACTIVITYMETRICS
--             table from the DB2ACTIVITIES event monitor definition.
--           - If used with a version higher than this, consult the DB2
--             documentation for any new enhancements to any of these event
--             monitors that may be of interest to you.
--


-- Create the statistics event monitor
CREATE EVENT MONITOR DB2STATISTICS
        FOR STATISTICS
        WRITE TO TABLE
              SCSTATS (TABLE SCSTATS_DB2STATISTICS IN TS_MONITORING),
              WCSTATS (TABLE WCSTATS_DB2STATISTICS IN TS_MONITORING),
              WLSTATS (TABLE WLSTATS_DB2STATISTICS IN TS_MONITORING),
              QSTATS (TABLE QSTATS_DB2STATISTICS IN TS_MONITORING),
              HISTOGRAMBIN (TABLE HISTOGRAMBIN_DB2STATISTICS IN TS_MONITORING),
              CONTROL (TABLE CONTROL_DB2STATISTICS IN TS_MONITORING);

-- Create the threshold violations event monitor
CREATE EVENT MONITOR DB2THRESHOLDS
        FOR THRESHOLD VIOLATIONS
        WRITE TO TABLE
              THRESHOLDVIOLATIONS (TABLE THRESHOLDVIOLATIONS_DB2THRESHOLDS
                                                      IN TS_MONITORING),
              CONTROL (TABLE CONTROL_DB2THRESHOLDVIOLATIONS IN TS_MONITORING);
```

```
-- Create the activity event monitor
CREATE EVENT MONITOR DB2ACTIVITIES
        FOR ACTIVITIES
        WRITE TO TABLE
        ACTIVITY (TABLE ACTIVITY_DB2ACTIVITIES IN TS_MONITORING),
        ACTIVITYMETRICS (TABLE ACTIVITYMETRICS_DB2ACTIVITIES IN TS_MONITORING),
        ACTIVITYSTMT (TABLE ACTIVITYSTMT_DB2ACTIVITIES IN TS_MONITORING),
        ACTIVITYVALS (TABLE ACTIVITYVALS_DB2ACTIVITIES IN TS_MONITORING),
        CONTROL (TABLE CONTROL_DB2ACTIVITIES in TS_MONITORING);
```

Appendix C. DDL scripts for transitioning from stage 0 to stage 1

Stage 1 template script

The following is an example script that you can use to transition your stage 0 default configuration to a stage 1 best practices template configuration (for an editable version of this SQL script, you can use the `sample_create_wlm_bp_template.sql` file included in the best practices `.zip` file):

```
--
-- Script of DDL statements needed to lay down initial template as prescribed by
-- the DB2 Workload Management Best Practices for Data Warehouses
---

-- Change to special administration workload for this connection while running
-- this script in order to bypass any existing workload management configuration
-- rules
SET WORKLOAD TO SYSDEFAULTADMWORKLOAD;

-- Create disabled superclass for workload management best practices for
-- warehouse template
CREATE SERVICE CLASS "WLMBP_MASTER" DISABLE;

-- Create disabled service subclass for COMPLEX DML statements
-- with extended aggregate data being gathered
CREATE SERVICE CLASS "COMPLEX_DML" UNDER "WLMBP_MASTER"
    COLLECT AGGREGATE ACTIVITY DATA EXTENDED DISABLE;

-- Create disabled service subclass for MEDIUM DML statements
-- with extended aggregate data being gathered
CREATE SERVICE CLASS "MEDIUM_DML" UNDER "WLMBP_MASTER"
    COLLECT AGGREGATE ACTIVITY DATA EXTENDED DISABLE;

-- Create disabled service subclass for SIMPLE DML statements
-- with extended aggregate data being gathered
CREATE SERVICE CLASS "SIMPLE_DML" UNDER "WLMBP_MASTER"
    COLLECT AGGREGATE ACTIVITY DATA EXTENDED DISABLE;

-- Create disabled service subclass for MINOR DML statements
-- with extended aggregate data being gathered
CREATE SERVICE CLASS "MINOR_DML" UNDER "WLMBP_MASTER"
    COLLECT AGGREGATE ACTIVITY DATA EXTENDED DISABLE;

-- Create disabled service subclass for TRIVIAL DML statements
-- with extended aggregate data being gathered
CREATE SERVICE CLASS "TRIVIAL_DML" UNDER "WLMBP_MASTER"
    COLLECT AGGREGATE ACTIVITY DATA EXTENDED DISABLE;

-- Create disabled service subclass for ETL work (e.g. LOAD)
-- with extended aggregate data being gathered
CREATE SERVICE CLASS "ETL" UNDER "WLMBP_MASTER"
    COLLECT AGGREGATE ACTIVITY DATA EXTENDED DISABLE;

-- Create set of work class definitions for incoming work
CREATE WORK CLASS SET "WLMBP_WORK_CLASSES"
(
-- Define as trivial cost, any DML statement
--  with estimated cost between 0 and 5,000 timerons
  WORK CLASS "TRIVIAL_COST_DML"
        WORK TYPE DML
        FOR TIMERONCOST FROM 0.0 TO 5000.0,

-- Define as minor cost, any DML statement
--  with estimated cost between 5,000 and 30,000 timerons
```

```
  WORK CLASS "MINOR_COST_DML"
        WORK TYPE DML
        FOR TIMERONCOST FROM 5000.0 TO 30000.0,

-- Define as simple cost, any DML statement
--  with estimated cost between 30,000 and 300,000 timerons
  WORK CLASS "SIMPLE_COST_DML"
        WORK TYPE DML
        FOR TIMERONCOST FROM 30000.0 TO 300000.0,

-- Define as medium cost, any DML statement
--  with estimated cost between 300,000 and 5,000,000 timerons
  WORK CLASS "MEDIUM_COST_DML"
        WORK TYPE DML
        FOR TIMERONCOST FROM 300000.0 TO 5000000.0,

-- Define as complex cost, any DML statement
--  with estimated cost above 5,000,000 timerons
  WORK CLASS "COMPLEX_COST_DML"
        WORK TYPE DML
        FOR TIMERONCOST FROM 5000000.0 TO UNBOUNDED,

-- Define as DDL, any DDL statement
  WORK CLASS "DDL"
        WORK TYPE DDL,

-- Define as Load, any invocation of the Load utility
  WORK CLASS "LOAD"
        WORK TYPE LOAD,

-- Define as Call, any CALL statement
  WORK CLASS "CALL"
        WORK TYPE CALL
);

-- Create disabled activitytotaltime threshold for trivial subclass
CREATE THRESHOLD "WLMBP_TRIVIAL_DML_TIMEOUT"
        FOR SERVICE CLASS "TRIVIAL_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN ACTIVITYTOTALTIME > 1 MINUTE
                COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
                CONTINUE;

-- Create disabled activitytotaltime threshold for minor subclass
CREATE THRESHOLD "WLMBP_MINOR_DML_TIMEOUT"
        FOR SERVICE CLASS "MINOR_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN ACTIVITYTOTALTIME > 1 MINUTE
                COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
                CONTINUE;

-- Create disabled activity concurrency threshold for minor subclass
CREATE THRESHOLD "WLMBP_MINOR_DML_CONCURRENCY"
        FOR SERVICE CLASS "MINOR_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN CONCURRENTDBCOORDACTIVITIES > 40 AND QUEUEDACTIVITIES UNBOUNDED
                CONTINUE;

-- Create disabled activitytotaltime threshold for simple subclass
CREATE THRESHOLD "WLMBP_SIMPLE_DML_TIMEOUT"
        FOR SERVICE CLASS "SIMPLE_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN ACTIVITYTOTALTIME > 5 MINUTES
                COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
                CONTINUE;

-- Create disabled activity concurrency threshold for simple subclass
```

```
CREATE THRESHOLD "WLMBP_SIMPLE_DML_CONCURRENCY"
        FOR SERVICE CLASS "SIMPLE_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN CONCURRENTDBCOORDACTIVITIES > 16 AND QUEUEDACTIVITIES UNBOUNDED
                CONTINUE;

-- Create disabled activitytotaltime threshold for medium subclass
CREATE THRESHOLD "WLMBP_MEDIUM_DML_TIMEOUT"
        FOR SERVICE CLASS "MEDIUM_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN ACTIVITYTOTALTIME > 1 HOUR
                COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
                CONTINUE;

-- Create disabled activity concurrency threshold for medium subclass
CREATE THRESHOLD "WLMBP_MEDIUM_DML_CONCURRENCY"
        FOR SERVICE CLASS "MEDIUM_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN CONCURRENTDBCOORDACTIVITIES > 8 AND QUEUEDACTIVITIES UNBOUNDED
                CONTINUE;

-- Create disabled activitytotaltime threshold for complex subclass
CREATE THRESHOLD "WLMBP_COMPLEX_DML_TIMEOUT"
        FOR SERVICE CLASS "COMPLEX_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN ACTIVITYTOTALTIME > 4 HOURS
                COLLECT ACTIVITY DATA ON COORDINATOR WITH DETAILS
                CONTINUE;

-- Create disabled activity concurrency threshold for complex subclass
CREATE THRESHOLD "WLMBP_COMPLEX_DML_CONCURRENCY"
        FOR SERVICE CLASS "COMPLEX_DML" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN CONCURRENTDBCOORDACTIVITIES > 4 AND QUEUEDACTIVITIES UNBOUNDED
                CONTINUE;

-- Create disabled activity concurrency threshold for ETL subclass
CREATE THRESHOLD "WLMBP_ETL_CONCURRENCY"
        FOR SERVICE CLASS "ETL" UNDER "WLMBP_MASTER"
        ACTIVITIES ENFORCEMENT DATABASE
                DISABLE
        WHEN CONCURRENTDBCOORDACTIVITIES > 4 AND QUEUEDACTIVITIES UNBOUNDED
                CONTINUE;

-- Create disabled work action set for template service superclass which maps
-- key work types to appropriate service subclass
CREATE WORK ACTION SET "WLMBP_WORK_ACTIONS"
        FOR SERVICE CLASS "WLMBP_MASTER"
        USING WORK CLASS SET "WLMBP_WORK_CLASSES"
(
-- Direct trivial cost DML to the Trivial DML subclass and allow any nested SQL
-- statement to be re-evaluated
  WORK ACTION "MAP_TRIVIAL_COST_DML"
        ON WORK CLASS "TRIVIAL_COST_DML"
        MAP ACTIVITY WITHOUT NESTED TO "TRIVIAL_DML",

-- Direct minor cost DML to the Minor DML subclass and allow any nested SQL
-- statement to be re-evaluated
  WORK ACTION "MAP_MINOR_COST_DML"
        ON WORK CLASS "MINOR_COST_DML"
        MAP ACTIVITY WITHOUT NESTED TO "MINOR_DML",

-- Direct simple cost DML to the Simple DML subclass and allow any nested SQL
-- statement to be re-evaluated
  WORK ACTION "MAP_SIMPLE_COST_DML"
        ON WORK CLASS "SIMPLE_COST_DML"
```

```
        MAP ACTIVITY WITHOUT NESTED TO "SIMPLE_DML",

-- Direct medium cost DML to the Medium DML subclass and allow any nested SQL
-- statement to be re-evaluated
  WORK ACTION "MAP_MEDIUM_COST_DML"
        ON WORK CLASS "MEDIUM_COST_DML"
        MAP ACTIVITY WITHOUT NESTED TO "MEDIUM_DML",

-- Direct complex cost DML to the Complex DML subclass and allow any nested SQL
-- statement to be re-evaluated
  WORK ACTION "MAP_COMPLEX_COST_DML"
        ON WORK CLASS "COMPLEX_COST_DML"
        MAP ACTIVITY WITHOUT NESTED TO "COMPLEX_DML",

-- Direct Loads to the ETL subclass and allow any nested SQL to be re-evaluated
  WORK ACTION "MAP_LOAD"
        ON WORK CLASS "LOAD"
        MAP ACTIVITY WITHOUT NESTED TO "ETL"

--
-- DDL and CALL statements as well as other requests not explicitly mentioned
-- above will fall through and execute in the default service subclass of the
-- WLMBP_MASTER service superclass
--
)
DISABLE;

-- Enable template service superclass
ALTER SERVICE CLASS "WLMBP_MASTER" ENABLE;

-- Enable Complex DML service subclass
ALTER SERVICE CLASS "COMPLEX_DML" UNDER "WLMBP_MASTER" ENABLE;

-- Enable Medium DML service subclass
ALTER SERVICE CLASS "MEDIUM_DML" UNDER "WLMBP_MASTER" ENABLE;

-- Enable simple DML service subclass
ALTER SERVICE CLASS "SIMPLE_DML" UNDER "WLMBP_MASTER" ENABLE;

-- Enable Minor DML service subclass
ALTER SERVICE CLASS "MINOR_DML" UNDER "WLMBP_MASTER" ENABLE;

-- Enable Trivial DML service subclass
ALTER SERVICE CLASS "TRIVIAL_DML" UNDER "WLMBP_MASTER" ENABLE;

-- Enable ETL service subclass
ALTER SERVICE CLASS "ETL" UNDER "WLMBP_MASTER" ENABLE;

-- Enable work action set for template service superclass
ALTER WORK ACTION SET "WLMBP_WORK_ACTIONS" ENABLE;

-- Enable the activitytotaltime thresholds
ALTER THRESHOLD "WLMBP_TRIVIAL_DML_TIMEOUT" ENABLE;
ALTER THRESHOLD "WLMBP_MINOR_DML_TIMEOUT" ENABLE;
ALTER THRESHOLD "WLMBP_SIMPLE_DML_TIMEOUT" ENABLE;
ALTER THRESHOLD "WLMBP_MEDIUM_DML_TIMEOUT" ENABLE;
ALTER THRESHOLD "WLMBP_COMPLEX_DML_TIMEOUT" ENABLE;


--
-- Optional: You can enable the concurrency thresholds now (by removing the
-- comment prefix "--" in front of each ALTER statement) or at a later time
--
-- ALTER THRESHOLD "WLMBP_MINOR_DML_CONCURRENCY" ENABLE;
-- ALTER THRESHOLD "WLMBP_SIMPLE_DML_CONCURRENCY" ENABLE;
-- ALTER THRESHOLD "WLMBP_MEDIUM_DML_CONCURRENCY" ENABLE;
-- ALTER THRESHOLD "WLMBP_COMPLEX_DML_CONCURRENCY" ENABLE;
-- ALTER THRESHOLD "WLMBP_ETL_CONCURRENCY" ENABLE;


--
```

```
-- ******************************************************************************
-- ******************************************************************************
-- The next statement activates the new workload management configuration for
-- use on your system by redirecting the standard default user workload to the
-- new workload management best practices template configuration created above.
--

-- Alter default user workload to point to template service superclass and to
-- gather extended aggregate activity data
ALTER WORKLOAD "SYSDEFAULTUSERWORKLOAD" SERVICE CLASS "WLMBP_MASTER"
        COLLECT AGGREGATE ACTIVITY DATA EXTENDED;

-- Change back to standard workload selection for this connection
SET WORKLOAD TO AUTOMATIC;
```

Stage 1 template drop script

The following is an example script that you can use to remove the stage 1 best practices template configuration (for an editable version of this SQL script, you can use the `sample_drop_wlm_bp_template.sql` file included in the best practices `.zip` file):

```
-- Script of DDL statements needed to remove initial template prescribed by
-- the DB2 Workload Management Best Practices for Data Warehouses
--
-- This script is designed to remove the original, unmodified template. If you
-- have made modifications to the template some statements may return errors if
-- template objects have been removed and you may need to add new statements if
-- new objects have been created.
--
--

-- Change to special administration workload for this connection while running
-- this script in order to bypass any existing workload management configuration
-- rules
SET WORKLOAD TO SYSDEFAULTADMWORKLOAD;

--
-- ******************************************************************************
-- ******************************************************************************
-- The next statement de-activates the workload management best practices
-- template configuration on your system by redirecting the standard default
-- user workload back to the original DB2 user service superclass


-- Alter default user workload to point to template service superclass and to
-- gather extended aggregate activity data
ALTER WORKLOAD "SYSDEFAULTUSERWORKLOAD" SERVICE CLASS "SYSDEFAULTUSERCLASS"
        COLLECT AGGREGATE ACTIVITY DATA NONE;


-- Drop all activitytotaltime thresholds
DROP THRESHOLD "WLMBP_TRIVIAL_DML_TIMEOUT";
DROP THRESHOLD "WLMBP_MINOR_DML_TIMEOUT";
DROP THRESHOLD "WLMBP_SIMPLE_DML_TIMEOUT";
DROP THRESHOLD "WLMBP_MEDIUM_DML_TIMEOUT";
DROP THRESHOLD "WLMBP_COMPLEX_DML_TIMEOUT";

-- Disable all concurrency thresholds
ALTER THRESHOLD "WLMBP_MINOR_DML_CONCURRENCY" DISABLE;
ALTER THRESHOLD "WLMBP_SIMPLE_DML_CONCURRENCY" DISABLE;
ALTER THRESHOLD "WLMBP_MEDIUM_DML_CONCURRENCY" DISABLE;
ALTER THRESHOLD "WLMBP_COMPLEX_DML_CONCURRENCY" DISABLE;
ALTER THRESHOLD "WLMBP_ETL_CONCURRENCY" DISABLE;

-- The following statements may return an error if there is still user work
-- queued in the concurrency threshhold being dropped. If this occurs, wait
-- until the work is no longer queued and try the statement again.

-- Drop all concurrency thresholds
```

```
DROP THRESHOLD "WLMBP_MINOR_DML_CONCURRENCY";
DROP THRESHOLD "WLMBP_SIMPLE_DML_CONCURRENCY";
DROP THRESHOLD "WLMBP_MEDIUM_DML_CONCURRENCY";
DROP THRESHOLD "WLMBP_COMPLEX_DML_CONCURRENCY";
DROP THRESHOLD "WLMBP_ETL_CONCURRENCY";

-- Drop template work action set
DROP WORK ACTION SET "WLMBP_WORK_ACTIONS";

-- Drop template work class set
DROP WORK CLASS SET "WLMBP_WORK_CLASSES";

-- Disbale and drop all service subclasses under template superclass
ALTER SERVICE CLASS "COMPLEX_DML" UNDER "WLMBP_MASTER" DISABLE;
ALTER SERVICE CLASS "MEDIUM_DML" UNDER "WLMBP_MASTER" DISABLE;
ALTER SERVICE CLASS "SIMPLE_DML" UNDER "WLMBP_MASTER" DISABLE;
ALTER SERVICE CLASS "MINOR_DML" UNDER "WLMBP_MASTER"DISABLE;
ALTER SERVICE CLASS "TRIVIAL_DML" UNDER "WLMBP_MASTER" DISABLE;
ALTER SERVICE CLASS "ETL" UNDER "WLMBP_MASTER" DISABLE;

-- The following statements may return an error if there is still user work
-- present in the service class being dropped. If this occurs, wait until the
-- work is complete and try the statement again.
DROP SERVICE CLASS "COMPLEX_DML" UNDER "WLMBP_MASTER";
DROP SERVICE CLASS "MEDIUM_DML" UNDER "WLMBP_MASTER";
DROP SERVICE CLASS "SIMPLE_DML" UNDER "WLMBP_MASTER";
DROP SERVICE CLASS "MINOR_DML" UNDER "WLMBP_MASTER";
DROP SERVICE CLASS "TRIVIAL_DML" UNDER "WLMBP_MASTER";
DROP SERVICE CLASS "ETL" UNDER "WLMBP_MASTER";

-- Disable and drop template service superclass
ALTER SERVICE CLASS "WLMBP_MASTER" DISABLE;
DROP SERVICE CLASS "WLMBP_MASTER";

-- Change back to standard workload selection for this connection
SET WORKLOAD TO AUTOMATIC;
```

Appendix D. Techniques for adjusting work class definitions

This appendix covers some techniques that can be used for evaluating and adjusting the estimated cost ranges used in the work class definitions provided in the best practices template configuration to better match the actual workload being placed upon the system.

For the work class definitions used in the best practices template, see "Template work class definitions" on page 19.

As a reminder, the objective of these steps is to review the distribution of estimated costs, gathered over time, for each service subclass with an eye to identifying boundary values and the distribution of work within that service subclass. With this information, the initial template values can be adjusted to help ensure that the overall objective of having relatively homogeneous groups of work (in terms of estimated cost) within each service subclass is achieved so that system performance volatility is reduced when under concurrency control.

Finally, since real-life is often much more complicated, the steps outlined in the following sections are provided simply as general guidelines on how you can ensure the relative uniformity of work within a service subclass. The goal is not to achieve perfection, but rather simplify and tweak the standard best practices template to help ensure a good fit for your actual database workload. There has to be balance between the number of service subclasses, and the granularity of control that they provide over the mix of work executing on the system, versus the increasing complexity of your workload management configuration.

The overriding philosophy to keep in mind while doing this exercise is to err on the side of not changing anything if it is not obvious what to do. If a situation is encountered which is not easily mapped to one of the following scenarios, remember that the effects of a less than desirable spread of estimated costs for work within a service subclass are not in themselves disastrous and might not be noticed on many systems.

This in-depth material supports the best practices documentation in "Adjusting work class definitions" on page 34.

Analyzing activity event monitor data

After you have collected data in the activity event monitor that represents all or a good subset of the overall work to be run on your system, it is possible to analyze this data. Analysis of the data will determine how the work will interact with the best practices template and what is the natural distribution of the estimated cost values for your workload.

An SQL query can be used to determine how the captured work maps to the target service subclasses in the predefined best practices template, as well as provide a relatively granular perspective of the estimated cost distribution. For a sample SQL script, see: "Sample D1: Determining estimated cost distribution based on activity event monitor data" on page 82.

It is possible to gather an estimated cost histogram from the data gathered by the statistics event monitor if the service classes have been set to use the EXTENDED

option of the COLLECT AGGREGATE ACTIVITY DATA clause. Although this data does not contain detailed information about individual entries, but does show the overall distribution of estimated costs for incoming SQL statements that execute in each service subclass. For a sample SQL script, see: "Sample D2: Determining estimated cost distribution based on statistics event monitor data" on page 84.

The following is a list of key things to remember about this data:

- The data represents the coordinator statement perspective and does not contain costs for nested SQL statements such as those invoked from within a stored procedure.
- The bins in the histogram information represent a logarithmic scale (higher bins represent much larger ranges than lower bins) and any analysis on that data must take this scale into account when adjusting service subclasses.
- If an insufficient distribution across the bins is discovered (all the values are falling into one bin), it is possible to adjust the template scale to provide better focus and distribution information. You can zoom the scale of the histogram template in or out to get the resolution that you want on the distribution information by using the CREATE TEMPLATE and ALTER TEMPLATE statements.

Sample D1: Determining estimated cost distribution based on activity event monitor data

This SQL query evaluates all the individual entries in the activity event monitor based on the estimated costs of each statement that was executed and, based on a predetermined set of ranges, reports on the counts in each range. This query only considers DML statements since those are the only ones with estimated costs provided by the SQL Compiler.

SQL text:

For an editable version of this SQL script, you can use the `sampleD1.sql` file included in the best practices `.zip` file.

```
--
-- Query to classify data captured in activity event monitor by best practices
-- work class definition and by estimated cost range.
--
WITH

-- Classify all DML statements captured by what class of DML statement they
-- would be considered by the workload management best practices. Also calculate
-- lifetime for each.
V1 AS
   (SELECT CASE
             WHEN QUERY_COST_ESTIMATE < 5000 THEN 'TRIVIAL'
             WHEN QUERY_COST_ESTIMATE BETWEEN 5000 AND 30000 THEN 'MINOR'
             WHEN QUERY_COST_ESTIMATE BETWEEN 30000 AND 300000 THEN 'SIMPLE'
             WHEN QUERY_COST_ESTIMATE BETWEEN 300000 AND 5000000 THEN 'MEDIUM'
             WHEN QUERY_COST_ESTIMATE > 5000000 THEN 'COMPLEX'
           END AS TARGET_WORKCLASS,
           QUERY_COST_ESTIMATE,
           (TIME_COMPLETED - TIME_STARTED) AS LIFETIME_DURATION
    FROM ACTIVITY_DB2ACTIVITIES
    WHERE ACTIVITY_TYPE IN ('READ_DML','WRITE_DML')
       AND PARTITION_NUMBER = COORD_PARTITION_NUM),

-- Identify the pre-determined cost range for each statement
 V2 AS
    (SELECT TARGET_WORKCLASS,
            CASE
              WHEN QUERY_COST_ESTIMATE < 1000 THEN '000K_TO_1K'
              WHEN QUERY_COST_ESTIMATE BETWEEN 1000
                                          AND 2000 THEN '001K_TO_2K'
              WHEN QUERY_COST_ESTIMATE BETWEEN 2000
                                          AND 3000 THEN '002K_TO_3K'
              WHEN QUERY_COST_ESTIMATE BETWEEN 3000
```

```
                                        AND 4000 THEN '003K_TO_4K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 4000
                                        AND 5000 THEN '004K_TO_5K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 5000
                                        AND 10000 THEN '005K_TO_10K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 10000
                                        AND 20000 THEN '010K_TO_20K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 20000
                                        AND 30000 THEN '020K_TO_30K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 30000
                                        AND 40000 THEN '030K_TO_40K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 40000
                                        AND 50000 THEN '040K_TO_50K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 50000
                                        AND 100000 THEN '050K_TO_100K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 100000
                                        AND 150000 THEN '100K_TO_150K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 150000
                                        AND 200000 THEN '150K_TO_200K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 200000
                                        AND 300000 THEN '200K_TO_300K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 300000
                                        AND 400000 THEN '300K_TO_400K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 400000
                                        AND 500000 THEN '400K_TO_500K'
                WHEN QUERY_COST_ESTIMATE BETWEEN 500000
                                        AND 1000000 THEN '500K_TO_1M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 1000000
                                        AND 2000000 THEN '001M_TO_2M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 2000000
                                        AND 3000000 THEN '002M_TO_3M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 3000000
                                        AND 4000000 THEN '003M_TO_4M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 4000000
                                        AND 5000000 THEN '004M_TO_5M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 5000000
                                        AND 6000000 THEN '005M_TO_6M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 6000000
                                        AND 7000000 THEN '006M_TO_7M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 7000000
                                        AND 8000000 THEN '007M_TO_8M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 8000000
                                        AND 9000000 THEN '008M_TO_9M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 9000000
                                        AND 10000000 THEN '009M_TO_10M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 10000000
                                        AND 20000000 THEN '010M_TO_20M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 20000000
                                        AND 30000000 THEN '020M_TO_30M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 30000000
                                        AND 40000000 THEN '030M_TO_40M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 40000000
                                        AND 50000000 THEN '040M_TO_50M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 50000000
                                        AND 60000000 THEN '050M_TO_60M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 60000000
                                        AND 70000000 THEN '060M_TO_70M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 70000000
                                        AND 80000000 THEN '070M_TO_80M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 80000000
                                        AND 90000000 THEN '080M_TO_90M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 90000000
                                        AND 100000000 THEN '090M_TO_100M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 100000000
                                        AND 500000000 THEN '100M_TO_500M'
                WHEN QUERY_COST_ESTIMATE BETWEEN 500000000
                                        AND 1000000000 THEN '500M_TO_1000M'
                WHEN QUERY_COST_ESTIMATE > 1000000000 THEN 'LARGER_THAN_1000M'
            END AS TIMERON_RANGE,
            QUERY_COST_ESTIMATE,
            LIFETIME_DURATION
        FROM V1)

-- Select the summary by work class with cost range distribution within that
-- class as well as min/max lifetime information
SELECT
  TARGET_WORKCLASS,
  TIMERON_RANGE,
  COUNT(*) AS COUNT,
```

```
  MIN(LIFETIME_DURATION) AS MIN_LIFETIME_DURATION,
  MAX(LIFETIME_DURATION) AS MAX_LIFETIME_DURATION
FROM V2
GROUP BY TARGET_WORKCLASS, TIMERON_RANGE
ORDER BY TARGET_WORKCLASS, TIMERON_RANGE;
```

Example output:

```
TARGET_WORKCLASS TIMERON_RANGE     COUNT       MIN_LIFETIME_DURATION  MAX_LIFETIME_DURATION
---------------- ---------------- ----------- ---------------------- ----------------------
MINOR            005K_TO_10K              530             101.932650            2644.595745
MINOR            010K_TO_20K              482             101.980664            1355.888548
MINOR            020K_TO_30K               86             101.843665            1246.850287
SIMPLE           030K_TO_40K                4            1219.134137            1752.144699
SIMPLE           040K_TO_50K                4             846.780529            1947.717285
SIMPLE           050K_TO_100K               9             502.923738             727.903279
SIMPLE           100K_TO_150K               2             600.377285             838.094614
SIMPLE           150K_TO_200K               1             534.224289             534.224289
TRIVIAL          000K_TO_1K              4060             100.367369            3457.374693
TRIVIAL          001K_TO_2K              1113             100.429077            3130.020908
TRIVIAL          002K_TO_3K               646             100.312853            3011.584925
TRIVIAL          003K_TO_4K               357             101.201110            4655.836927
TRIVIAL          004K_TO_5K               227             102.513602            2827.666814

  13 record(s) selected.
```

Sample D2: Determining estimated cost distribution based on statistics event monitor data

This SQL query evaluates all the individual entries in the activity event monitor based on the estimated costs of each statement that was executed and, based on a predetermined set of ranges, reports on the counts in each range. This query only considers DML statements since those are the only ones with estimated costs provided by the SQL Compiler.

SQL text:

For an editable version of this SQL script, you can use the `sampleD2.sql` file included in the best practices `.zip` file.

```
--
-- This query  produces a readable summary of the estimated cost distribution
-- data available for each service class collecting aggregate activity data.
-- The data is merged across all collections to produce an overall summary.
--
SELECT SUBSTR(PARENTSERVICECLASSNAME,1,32) AS SERVICE_SUPERCLASS_NAME,
       SUBSTR(SERVICECLASSNAME,1,32) AS SERVICE_SUBCLASS_NAME,
       BIN_ID,
       MIN(BOTTOM) AS BIN_BOTTOM_COST,
       MAX(TOP) AS BIN_TOP_COST,
       SUM(NUMBER_IN_BIN) AS BIN_COUNT
FROM HISTOGRAMBIN_DB2STATISTICS AS H, SYSCAT.SERVICECLASSES AS S
WHERE H.SERVICE_CLASS_ID = S.SERVICECLASSID
   AND H.SERVICE_CLASS_ID != 0
   AND HISTOGRAM_TYPE IN ('CoordActEstCost')
GROUP BY PARENTSERVICECLASSNAME, SERVICECLASSNAME, BIN_ID
ORDER BY PARENTSERVICECLASSNAME, SERVICECLASSNAME, BIN_ID;
```

Example output:

```
SERVICE_SUPERCLASS_NAME          SERVICE_SUBCLASS_NAME            BIN_ID      BIN_BOTTOM_COST      BIN_TOP_COST         BIN_COUNT
-------------------------------- -------------------------------- ----------- -------------------- -------------------- --------------------
WLMBP_MASTER                     COMPLEX_DML                                1                    0                    1                    0
WLMBP_MASTER                     COMPLEX_DML                                2                    1                    2                    0
WLMBP_MASTER                     COMPLEX_DML                                3                    2                    3                    0
WLMBP_MASTER                     COMPLEX_DML                                4                    3                    5                    0
WLMBP_MASTER                     COMPLEX_DML                                5                    5                    8                    0
WLMBP_MASTER                     COMPLEX_DML                                6                    8                   12                    0
WLMBP_MASTER                     COMPLEX_DML                                7                   12                   19                    0
WLMBP_MASTER                     COMPLEX_DML                                8                   19                   29                    0
WLMBP_MASTER                     COMPLEX_DML                                9                   29                   44                    0
WLMBP_MASTER                     COMPLEX_DML                               10                   44                   68                    0
WLMBP_MASTER                     COMPLEX_DML                               11                   68                  103                    0
WLMBP_MASTER                     COMPLEX_DML                               12                  103                  158                    0
WLMBP_MASTER                     COMPLEX_DML                               13                  158                  241                    0
WLMBP_MASTER                     COMPLEX_DML                               14                  241                  369                    0
WLMBP_MASTER                     COMPLEX_DML                               15                  369                  562                    0
WLMBP_MASTER                     COMPLEX_DML                               16                  562                  858                    0
WLMBP_MASTER                     COMPLEX_DML                               17                  858                 1309                    0
WLMBP_MASTER                     COMPLEX_DML                               18                 1309                 1997                    0
WLMBP_MASTER                     COMPLEX_DML                               19                 1997                 3046                    0
```

WLMBP_MASTER	COMPLEX_DML	20	3046	4647	0
WLMBP_MASTER	COMPLEX_DML	21	4647	7089	0
WLMBP_MASTER	COMPLEX_DML	22	7089	10813	0
WLMBP_MASTER	COMPLEX_DML	23	10813	16493	0
WLMBP_MASTER	COMPLEX_DML	24	16493	25157	0
WLMBP_MASTER	COMPLEX_DML	25	25157	38373	0
WLMBP_MASTER	COMPLEX_DML	26	38373	58532	0
WLMBP_MASTER	COMPLEX_DML	27	58532	89280	0
WLMBP_MASTER	COMPLEX_DML	28	89280	136181	0
WLMBP_MASTER	COMPLEX_DML	29	136181	207720	0
WLMBP_MASTER	COMPLEX_DML	30	207720	316840	0
WLMBP_MASTER	COMPLEX_DML	31	316840	483283	0
WLMBP_MASTER	COMPLEX_DML	32	483283	737162	0
WLMBP_MASTER	COMPLEX_DML	33	737162	1124409	0
WLMBP_MASTER	COMPLEX_DML	34	1124409	1715085	0
WLMBP_MASTER	COMPLEX_DML	35	1715085	2616055	0
WLMBP_MASTER	COMPLEX_DML	36	2616055	3990325	0
WLMBP_MASTER	COMPLEX_DML	37	3990325	6086529	0
WLMBP_MASTER	COMPLEX_DML	38	6086529	9283913	37
WLMBP_MASTER	COMPLEX_DML	39	9283913	14160950	0
WLMBP_MASTER	COMPLEX_DML	40	14160950	21600000	1
WLMBP_MASTER	COMPLEX_DML	41	21600000	-1	303
WLMBP_MASTER	ETL	1	0	1	0
WLMBP_MASTER	ETL	2	1	2	0
WLMBP_MASTER	ETL	3	2	3	0
WLMBP_MASTER	ETL	4	3	5	0
WLMBP_MASTER	ETL	5	5	8	0
WLMBP_MASTER	ETL	6	8	12	0
WLMBP_MASTER	ETL	7	12	19	0
WLMBP_MASTER	ETL	8	19	29	0
WLMBP_MASTER	ETL	9	29	44	0
WLMBP_MASTER	ETL	10	44	68	0
WLMBP_MASTER	ETL	11	68	103	0
WLMBP_MASTER	ETL	12	103	158	0
WLMBP_MASTER	ETL	13	158	241	0
WLMBP_MASTER	ETL	14	241	369	0
WLMBP_MASTER	ETL	15	369	562	0
WLMBP_MASTER	ETL	16	562	858	0
WLMBP_MASTER	ETL	17	858	1309	0
WLMBP_MASTER	ETL	18	1309	1997	0
WLMBP_MASTER	ETL	19	1997	3046	0
WLMBP_MASTER	ETL	20	3046	4647	0
WLMBP_MASTER	ETL	21	4647	7089	0
WLMBP_MASTER	ETL	22	7089	10813	0
WLMBP_MASTER	ETL	23	10813	16493	0
WLMBP_MASTER	ETL	24	16493	25157	0
WLMBP_MASTER	ETL	25	25157	38373	0
WLMBP_MASTER	ETL	26	38373	58532	0
WLMBP_MASTER	ETL	27	58532	89280	0
WLMBP_MASTER	ETL	28	89280	136181	0
WLMBP_MASTER	ETL	29	136181	207720	0
WLMBP_MASTER	ETL	30	207720	316840	0
WLMBP_MASTER	ETL	31	316840	483283	0
WLMBP_MASTER	ETL	32	483283	737162	0
WLMBP_MASTER	ETL	33	737162	1124409	0
WLMBP_MASTER	ETL	34	1124409	1715085	0
WLMBP_MASTER	ETL	35	1715085	2616055	0
WLMBP_MASTER	ETL	36	2616055	3990325	0
WLMBP_MASTER	ETL	37	3990325	6086529	0
WLMBP_MASTER	ETL	38	6086529	9283913	0
WLMBP_MASTER	ETL	39	9283913	14160950	0
WLMBP_MASTER	ETL	40	14160950	21600000	0
WLMBP_MASTER	ETL	41	21600000	-1	0
WLMBP_MASTER	MEDIUM_DML	1	0	1	0
WLMBP_MASTER	MEDIUM_DML	2	1	2	0
WLMBP_MASTER	MEDIUM_DML	3	2	3	0
WLMBP_MASTER	MEDIUM_DML	4	3	5	0
WLMBP_MASTER	MEDIUM_DML	5	5	8	0
WLMBP_MASTER	MEDIUM_DML	6	8	12	0
WLMBP_MASTER	MEDIUM_DML	7	12	19	0
WLMBP_MASTER	MEDIUM_DML	8	19	29	0
WLMBP_MASTER	MEDIUM_DML	9	29	44	0
WLMBP_MASTER	MEDIUM_DML	10	44	68	0
WLMBP_MASTER	MEDIUM_DML	11	68	103	0
WLMBP_MASTER	MEDIUM_DML	12	103	158	0
WLMBP_MASTER	MEDIUM_DML	13	158	241	0
WLMBP_MASTER	MEDIUM_DML	14	241	369	0
WLMBP_MASTER	MEDIUM_DML	15	369	562	0
WLMBP_MASTER	MEDIUM_DML	16	562	858	0
WLMBP_MASTER	MEDIUM_DML	17	858	1309	0
WLMBP_MASTER	MEDIUM_DML	18	1309	1997	0
WLMBP_MASTER	MEDIUM_DML	19	1997	3046	0
WLMBP_MASTER	MEDIUM_DML	20	3046	4647	0
WLMBP_MASTER	MEDIUM_DML	21	4647	7089	0
WLMBP_MASTER	MEDIUM_DML	22	7089	10813	0
WLMBP_MASTER	MEDIUM_DML	23	10813	16493	0
WLMBP_MASTER	MEDIUM_DML	24	16493	25157	0
WLMBP_MASTER	MEDIUM_DML	25	25157	38373	0
WLMBP_MASTER	MEDIUM_DML	26	38373	58532	0
WLMBP_MASTER	MEDIUM_DML	27	58532	89280	0

WLMBP_MASTER	MEDIUM_DML	28	89280	136181	0
WLMBP_MASTER	MEDIUM_DML	29	136181	207720	0
WLMBP_MASTER	MEDIUM_DML	30	207720	316840	11
WLMBP_MASTER	MEDIUM_DML	31	316840	483283	160
WLMBP_MASTER	MEDIUM_DML	32	483283	737162	43
WLMBP_MASTER	MEDIUM_DML	33	737162	1124409	65
WLMBP_MASTER	MEDIUM_DML	34	1124409	1715085	25
WLMBP_MASTER	MEDIUM_DML	35	1715085	2616055	12
WLMBP_MASTER	MEDIUM_DML	36	2616055	3990325	1
WLMBP_MASTER	MEDIUM_DML	37	3990325	6086529	23
WLMBP_MASTER	MEDIUM_DML	38	6086529	9283913	0
WLMBP_MASTER	MEDIUM_DML	39	9283913	14160950	0
WLMBP_MASTER	MEDIUM_DML	40	14160950	21600000	0
WLMBP_MASTER	MEDIUM_DML	41	21600000	-1	0
WLMBP_MASTER	MINOR_DML	1	0	1	0
WLMBP_MASTER	MINOR_DML	2	1	2	0
WLMBP_MASTER	MINOR_DML	3	2	3	0
WLMBP_MASTER	MINOR_DML	4	3	5	0
WLMBP_MASTER	MINOR_DML	5	5	8	0
WLMBP_MASTER	MINOR_DML	6	8	12	0
WLMBP_MASTER	MINOR_DML	7	12	19	0
WLMBP_MASTER	MINOR_DML	8	19	29	0
WLMBP_MASTER	MINOR_DML	9	29	44	0
WLMBP_MASTER	MINOR_DML	10	44	68	0
WLMBP_MASTER	MINOR_DML	11	68	103	0
WLMBP_MASTER	MINOR_DML	12	103	158	0
WLMBP_MASTER	MINOR_DML	13	158	241	0
WLMBP_MASTER	MINOR_DML	14	241	369	0
WLMBP_MASTER	MINOR_DML	15	369	562	0
WLMBP_MASTER	MINOR_DML	16	562	858	0
WLMBP_MASTER	MINOR_DML	17	858	1309	0
WLMBP_MASTER	MINOR_DML	18	1309	1997	0
WLMBP_MASTER	MINOR_DML	19	1997	3046	0
WLMBP_MASTER	MINOR_DML	20	3046	4647	0
WLMBP_MASTER	MINOR_DML	21	4647	7089	8128
WLMBP_MASTER	MINOR_DML	22	7089	10813	7868
WLMBP_MASTER	MINOR_DML	23	10813	16493	4910
WLMBP_MASTER	MINOR_DML	24	16493	25157	2042
WLMBP_MASTER	MINOR_DML	25	25157	38373	312
WLMBP_MASTER	MINOR_DML	26	38373	58532	0
WLMBP_MASTER	MINOR_DML	27	58532	89280	0
WLMBP_MASTER	MINOR_DML	28	89280	136181	0
WLMBP_MASTER	MINOR_DML	29	136181	207720	0
WLMBP_MASTER	MINOR_DML	30	207720	316840	0
WLMBP_MASTER	MINOR_DML	31	316840	483283	0
WLMBP_MASTER	MINOR_DML	32	483283	737162	0
WLMBP_MASTER	MINOR_DML	33	737162	1124409	0
WLMBP_MASTER	MINOR_DML	34	1124409	1715085	0
WLMBP_MASTER	MINOR_DML	35	1715085	2616055	0
WLMBP_MASTER	MINOR_DML	36	2616055	3990325	0
WLMBP_MASTER	MINOR_DML	37	3990325	6086529	0
WLMBP_MASTER	MINOR_DML	38	6086529	9283913	0
WLMBP_MASTER	MINOR_DML	39	9283913	14160950	0
WLMBP_MASTER	MINOR_DML	40	14160950	21600000	0
WLMBP_MASTER	MINOR_DML	41	21600000	-1	0
WLMBP_MASTER	SIMPLE_DML	1	0	1	0
WLMBP_MASTER	SIMPLE_DML	2	1	2	0
WLMBP_MASTER	SIMPLE_DML	3	2	3	0
WLMBP_MASTER	SIMPLE_DML	4	3	5	0
WLMBP_MASTER	SIMPLE_DML	5	5	8	0
WLMBP_MASTER	SIMPLE_DML	6	8	12	0
WLMBP_MASTER	SIMPLE_DML	7	12	19	0
WLMBP_MASTER	SIMPLE_DML	8	19	29	0
WLMBP_MASTER	SIMPLE_DML	9	29	44	0
WLMBP_MASTER	SIMPLE_DML	10	44	68	0
WLMBP_MASTER	SIMPLE_DML	11	68	103	0
WLMBP_MASTER	SIMPLE_DML	12	103	158	0
WLMBP_MASTER	SIMPLE_DML	13	158	241	0
WLMBP_MASTER	SIMPLE_DML	14	241	369	0
WLMBP_MASTER	SIMPLE_DML	15	369	562	0
WLMBP_MASTER	SIMPLE_DML	16	562	858	0
WLMBP_MASTER	SIMPLE_DML	17	858	1309	0
WLMBP_MASTER	SIMPLE_DML	18	1309	1997	0
WLMBP_MASTER	SIMPLE_DML	19	1997	3046	0
WLMBP_MASTER	SIMPLE_DML	20	3046	4647	0
WLMBP_MASTER	SIMPLE_DML	21	4647	7089	0
WLMBP_MASTER	SIMPLE_DML	22	7089	10813	0
WLMBP_MASTER	SIMPLE_DML	23	10813	16493	0
WLMBP_MASTER	SIMPLE_DML	24	16493	25157	0
WLMBP_MASTER	SIMPLE_DML	25	25157	38373	183
WLMBP_MASTER	SIMPLE_DML	26	38373	58532	304
WLMBP_MASTER	SIMPLE_DML	27	58532	89280	449
WLMBP_MASTER	SIMPLE_DML	28	89280	136181	108
WLMBP_MASTER	SIMPLE_DML	29	136181	207720	138
WLMBP_MASTER	SIMPLE_DML	30	207720	316840	77
WLMBP_MASTER	SIMPLE_DML	31	316840	483283	0
WLMBP_MASTER	SIMPLE_DML	32	483283	737162	0
WLMBP_MASTER	SIMPLE_DML	33	737162	1124409	0
WLMBP_MASTER	SIMPLE_DML	34	1124409	1715085	0
WLMBP_MASTER	SIMPLE_DML	35	1715085	2616055	0

```
WLMBP_MASTER                SIMPLE_DML                  36          2616055          3990325                0
WLMBP_MASTER                SIMPLE_DML                  37          3990325          6086529                0
WLMBP_MASTER                SIMPLE_DML                  38          6086529          9283913                0
WLMBP_MASTER                SIMPLE_DML                  39          9283913         14160950                0
WLMBP_MASTER                SIMPLE_DML                  40         14160950         21600000                0
WLMBP_MASTER                SIMPLE_DML                  41         21600000               -1                0
WLMBP_MASTER                TRIVIAL_DML                  1                0                1          7780946
WLMBP_MASTER                TRIVIAL_DML                  2                1                2           291323
WLMBP_MASTER                TRIVIAL_DML                  3                2                3            29700
WLMBP_MASTER                TRIVIAL_DML                  4                3                5                0
WLMBP_MASTER                TRIVIAL_DML                  5                5                8           899511
WLMBP_MASTER                TRIVIAL_DML                  6                8               12           736615
WLMBP_MASTER                TRIVIAL_DML                  7               12               19           692754
WLMBP_MASTER                TRIVIAL_DML                  8               19               29          1687141
WLMBP_MASTER                TRIVIAL_DML                  9               29               44           596125
WLMBP_MASTER                TRIVIAL_DML                 10               44               68           472569
WLMBP_MASTER                TRIVIAL_DML                 11               68              103           143499
WLMBP_MASTER                TRIVIAL_DML                 12              103              158           155808
WLMBP_MASTER                TRIVIAL_DML                 13              158              241            77685
WLMBP_MASTER                TRIVIAL_DML                 14              241              369            34033
WLMBP_MASTER                TRIVIAL_DML                 15              369              562            45917
WLMBP_MASTER                TRIVIAL_DML                 16              562              858            25083
WLMBP_MASTER                TRIVIAL_DML                 17              858             1309            20389
WLMBP_MASTER                TRIVIAL_DML                 18             1309             1997            20533
WLMBP_MASTER                TRIVIAL_DML                 19             1997             3046            16640
WLMBP_MASTER                TRIVIAL_DML                 20             3046             4647            11319
WLMBP_MASTER                TRIVIAL_DML                 21             4647             7089             1198
WLMBP_MASTER                TRIVIAL_DML                 22             7089            10813                0
WLMBP_MASTER                TRIVIAL_DML                 23            10813            16493                0
WLMBP_MASTER                TRIVIAL_DML                 24            16493            25157                0
WLMBP_MASTER                TRIVIAL_DML                 25            25157            38373                0
WLMBP_MASTER                TRIVIAL_DML                 26            38373            58532                0
WLMBP_MASTER                TRIVIAL_DML                 27            58532            89280                0
WLMBP_MASTER                TRIVIAL_DML                 28            89280           136181                0
WLMBP_MASTER                TRIVIAL_DML                 29           136181           207720                0
WLMBP_MASTER                TRIVIAL_DML                 30           207720           316840                0
WLMBP_MASTER                TRIVIAL_DML                 31           316840           483283                0
WLMBP_MASTER                TRIVIAL_DML                 32           483283           737162                0
WLMBP_MASTER                TRIVIAL_DML                 33           737162          1124409                0
WLMBP_MASTER                TRIVIAL_DML                 34          1124409          1715085                0
WLMBP_MASTER                TRIVIAL_DML                 35          1715085          2616055                0
WLMBP_MASTER                TRIVIAL_DML                 36          2616055          3990325                0
WLMBP_MASTER                TRIVIAL_DML                 37          3990325          6086529                0
WLMBP_MASTER                TRIVIAL_DML                 38          6086529          9283913                0
WLMBP_MASTER                TRIVIAL_DML                 39          9283913         14160950                0
WLMBP_MASTER                TRIVIAL_DML                 40         14160950         21600000                0
WLMBP_MASTER                TRIVIAL_DML                 41         21600000               -1                0

  246 record(s) selected.
```

A lumpy distribution

In most cases, you will discover that the distribution of work across the service subclasses, introduced by the best practices template, is widely and unevenly spread.

As mentioned in the opening section of this appendix, if none of the scenarios highlighted in the following sections are applicable to your distribution of estimated costs for work such as depicted in Figure 16 on page 88, it is acceptable to not adjust the default mapping behavior of the template and proceed to the next step in the overall process.

In the example depicted in Figure 16 on page 88, changes to the work classes and service classes are not necessary.

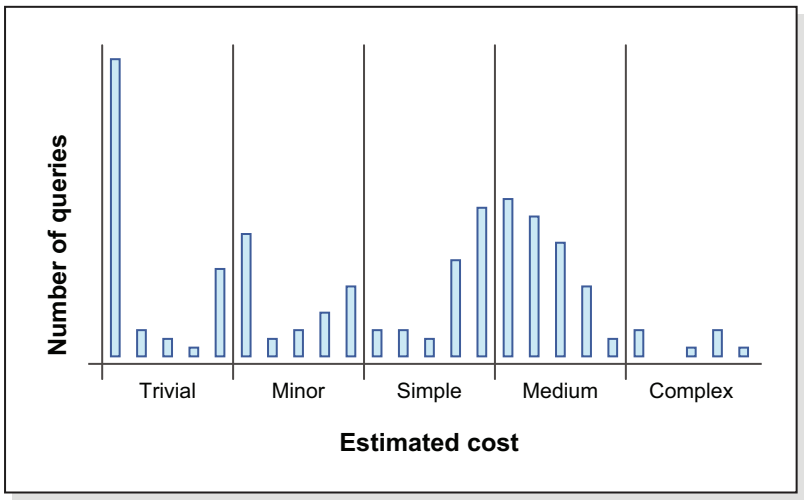


Figure 16. Lumpy distribution of estimated costs for work

No entries in a service subclass

If entries are not found in a service subclass, then remove the work class definition and its associated service subclass by expanding the estimated cost boundary and concurrency thresholds of the next, more expensive work class to include this class. The rationale for this action is that it is better to have more control imposed rather than less. The best practices configuration paradigm asserts more control on heavier work.

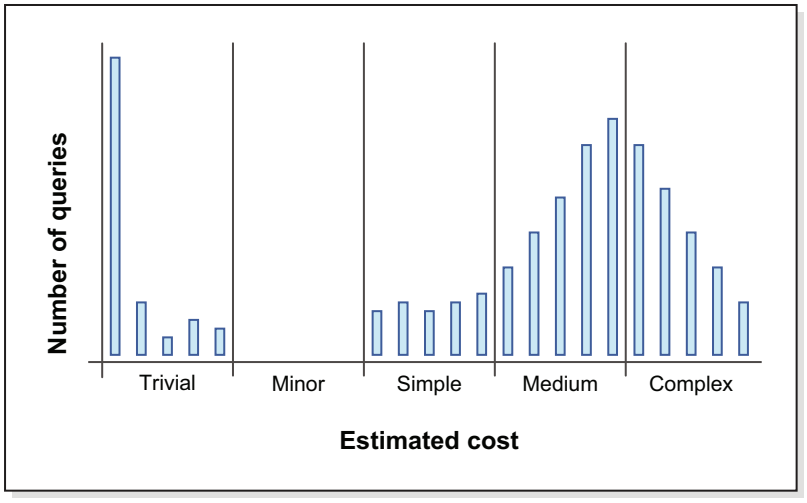


Figure 17. Empty service subclass distribution of estimated costs for work

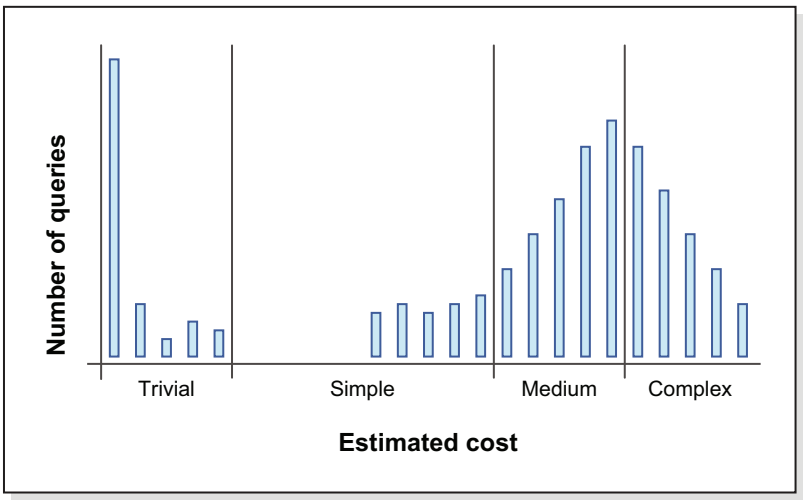


Figure 18. Modified empty service subclass distribution of estimated costs for work

For example, if queries were not found in the MINOR_DML service subclass, do the following:

1. Adjust the work action set
   - Remove the MAP_MINOR_COST_DML work action
2. Adjust the work class set
   - Remove the MINOR_COST_DML work class
   - Modify the lower estimated cost boundary of the SIMPLE_COST_DML work class to be equal to the lower boundary of the former MINOR_COST_DML work class
3. Adjust the DB2 concurrency thresholds
   - Add the value of the concurrency threshold on MINOR_DML service subclass to the value of the concurrency threshold in the SIMPLE_DML service subclass[6]
4. Adjust the DB2 service subclasses
   - Remove all activity and concurrency thresholds defined on the MINOR_DML service subclass
   - Remove the MINOR_DML service subclass

Minimal entries in a service subclass

If there are minimal entries in a service subclass, consider removing the service subclass in a manner similar to that described in the preceding section, except that you have the option of where to merge the entries in the class depending on where the few entries exist.

You also have the option of simply leaving the service subclass as it is. This option is never an incorrect decision.

6. We add the concurrency values at this point because we are still approaching this from the perspective of an initial concurrency *budget* and we have just merged the two divisions into one. Later on, we will look at adjusting the concurrency values based on actual resource consumption and response times.

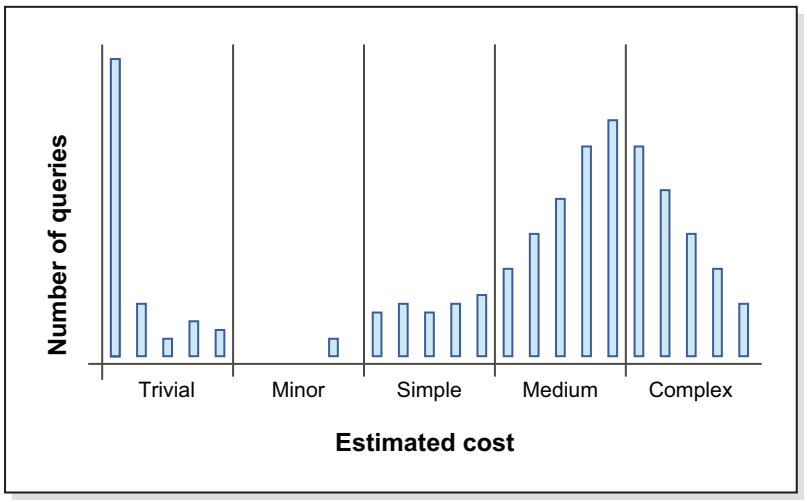


Figure 19. Minimal service subclass distribution of estimated costs for work

If you want to collapse service classes, then perform one of the following general rules of thumb:

- If the few entries exist only towards the upper end of the class as depicted in Figure 19, merge the class up as in the previous example.
- If the entries exist towards the lower end of the class, then merge downwards.
- If they exist in both the upper and lower ends as depicted in Figure 20, then follow the same approach as described in the next section dealing with U-shaped distributions.

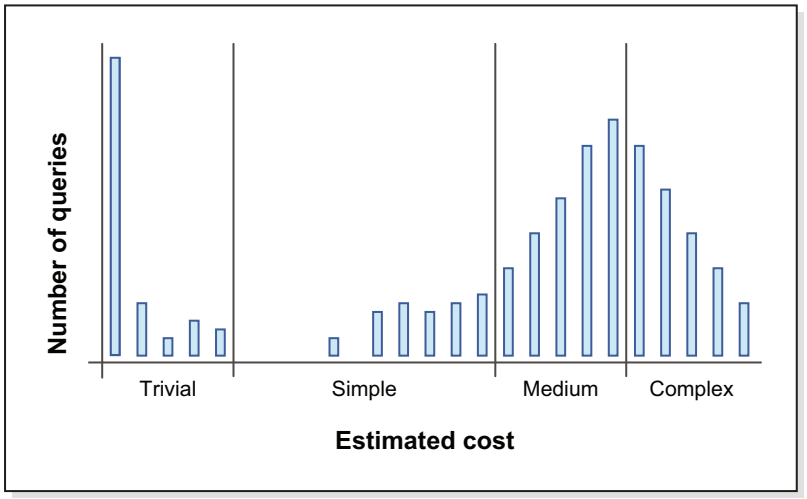


Figure 20. Modified minimal service subclass distribution of estimated costs for work

U-shaped distribution in a service subclass

This scenario is more difficult to determine as there is not always a clean gap between population centers. As before, simply leaving the service subclass alone is also a valid choice.

If the distribution of entries in a service subclass appears to have two or more distinct population centers around which other values cluster, then you are dealing

with a U-shaped distribution as depicted in Figure 21. Distinct populations in this context means that the two population centers have at least a magnitude of difference in their cost estimates and with few queries, if any, with cost estimates falling between these centers. There is a significant gap between the two clusters.

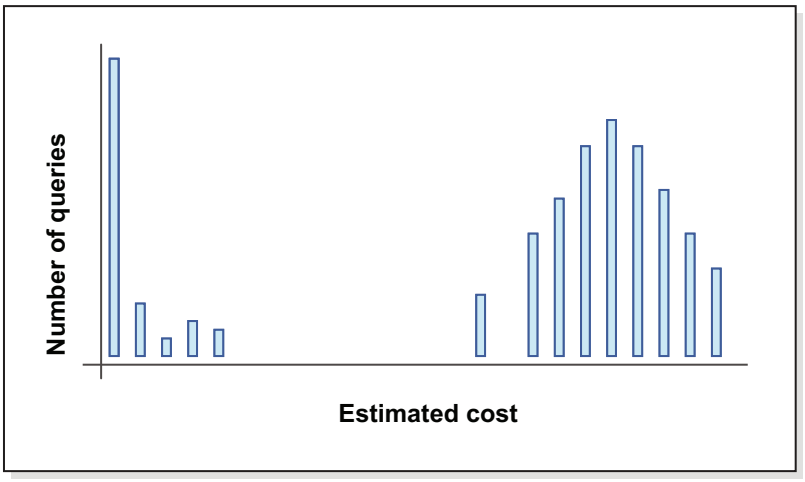


Figure 21. U-shaped distribution of estimated costs for work

The usual recommended treatment is to either adjust the class definition to reduce the number of centers in this class to one by moving one of them into a different class, or to remove the class completely by moving both centers to other service subclasses. A third option is to introduce a new subclass for one of the population centers.

The rationale for removing any *split* between the population centers is that concurrency thresholds are most effective in producing a stable, predictable system when the individual pieces of work that they control have a roughly equivalent impact on the system. The aim is to have stable and predictable demand from that service subclass. Having what would essentially be two different types of queries (in terms of resource consumption) in the same service subclass could result in an uneven and unpredictable impact on the system resources. This effect is more noticeable in the service subclasses with more costly work within them.

If the service class is merged with another existing one or a new service subclass is introduced, it is necessary to divide the concurrency allocation for the original service subclass with the other affected service classes. The best way to do this division is to divide the percentage allocated for that class between the other two classes by using some proportional means based on the population being moved, and then recalculate the concurrency values as was first done in "Allocating capacity" on page 30.

A general rule of thumb on what action might be appropriate for a U-shaped distribution, based on the service subclass in which it occurs, is something like the following:

- TRIVIAL_DML: Leave as-is
- MINOR_DML: Leave as-is, or merge
- SIMPLE_DML: Merge
- MEDIUM_DML: Introduce a new service subclass

- COMPLEX_DML: Introduce a new service subclass

Scenario 1

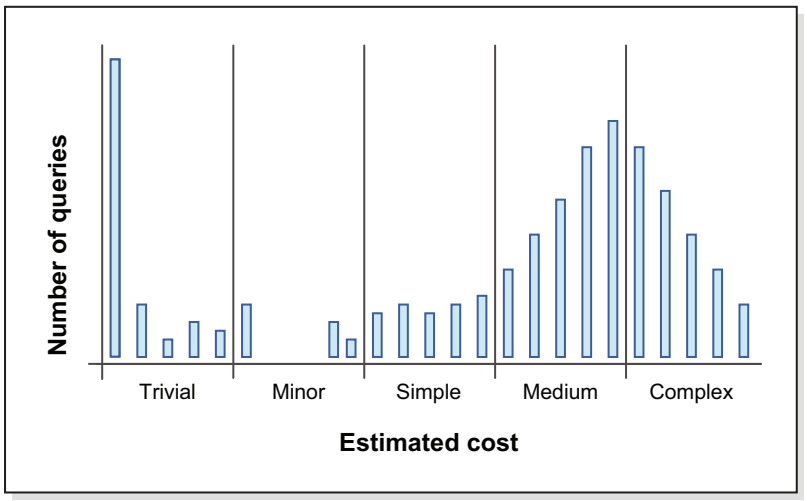


Figure 22. Scenario 1: U-shaped distribution of estimated costs for work

For example, if a number of queries are found clustered in the bottom and top of the MINOR_DML service subclass as depicted in Figure 22, do the following[7]:

1. Adjust the work action set
   a. Remove the MAP_MINOR_COST_DML work action
2. Adjust the work class set
   a. Remove the MINOR_COST_DML work class
   b. Modify the upper estimated cost boundary of the TRIVIAL_COST_DML work class to be equal to the median value of the (former) MINOR_COST_DML work class
   c. Modify the lower estimated cost boundary of the SIMPLE_COST_DML work class to be equal to the median value of the (former) MINOR_COST_DML work class
3. Adjust the DB2 concurrency thresholds
   a. Because the TRIVIAL_DML service subclass has no concurrency threshold in place initially and the number of queries is small, add the full value of the MINOR_DML service subclass concurrency threshold to the value of the concurrency threshold in the SIMPLE_DML service subclass
4. Adjust the DB2 service subclasses
   a. Remove all activity and concurrency thresholds defined on the MINOR_DML service subclass
   b. Remove the MINOR_DML service subclass as depicted in Figure 23 on page 93

7. Note that it is equally valid to approach this situation and implement the solution outlined in the next section where you leave the MINOR_DML service subclass to represent one of the two population centers and absorb the other into the adjacent service subclass.

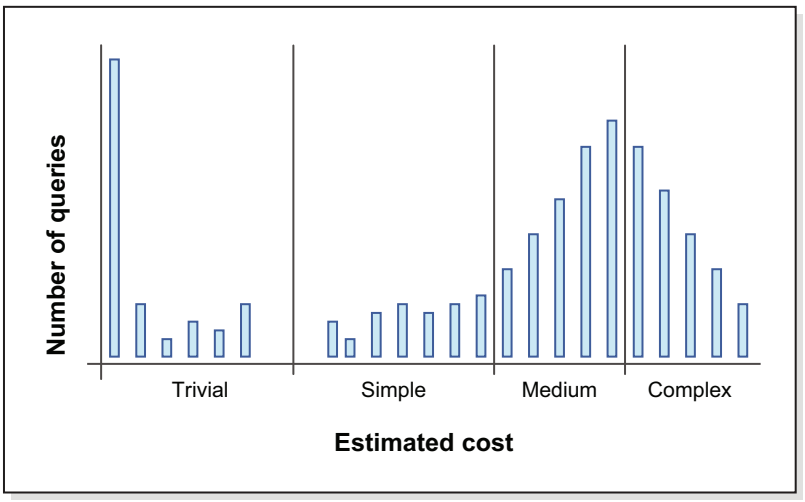


Figure 23. Scenario 1: Modified U-shaped distribution of estimated costs for work

Scenario 2

If one of the population centers is large and the other small as depicted in Figure 24, then it can be beneficial to merge the small cluster into surrounding service subclasses and leave this class solely for the larger population center.

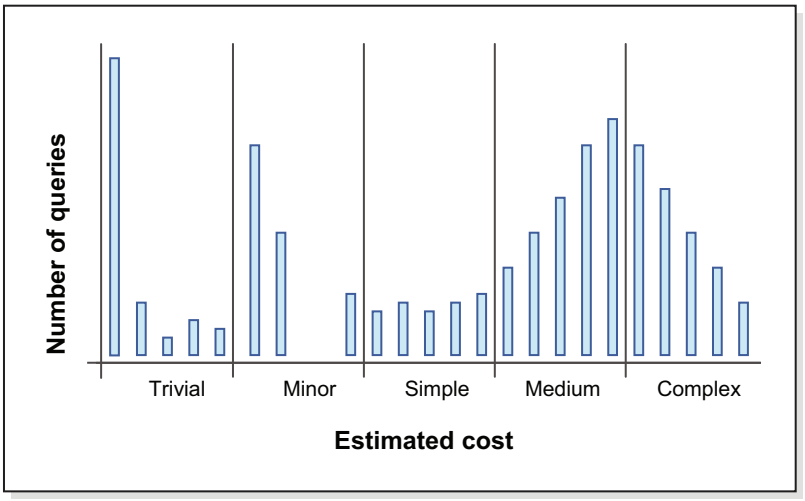


Figure 24. Scenario 2: U-shaped distribution of estimated costs for work

For example, if two such population centers (one large and one small) are found in the MINOR_DML service subclass with the smaller population center being near the upper end, do the following:

1. Adjust the work class set as depicted in Figure 25 on page 94
   a. Modify the upper estimated cost boundary of the MINOR_COST_DML work class to be less than the lowest value of the smaller population center at the upper end of the range
   b. Modify the lower estimated cost boundary of the SIMPLE_COST_DML work class to be equal to the same value chosen in the previous step
2. Adjust the DB2 concurrency thresholds

a. Determine the proportion of the work being moved from the MINOR_DML service subclass to the SIMPLE_DML service subclass, based on population ratio, and adjust the initial budget allocation to reflect the movement of this amount from the MINOR_DML to the SIMPLE_DML service subclass.
   1) Assume 10% movement of the population
   2) Initial concurrency budget allocation for MINOR_DML was 50% and SIMPLE_DML was 20%
   3) Adjusted concurrency budget allocation for MINOR_DML is 45% and SIMPLE_DML is 25%
   4) Initial concurrency threshold value for MINOR_DML was 40 and SIMPLE_DML was 16
   5) Adjusted concurrency threshold value for MINOR_DML is 36 and SIMPLE_DML is 20

b. Adjust the concurrency threshold on the MINOR_DML service subclass to the adjusted value (36) for that class

c. Adjust the concurrency threshold on the SIMPLE_DML service subclass to the adjusted value (20) for that class

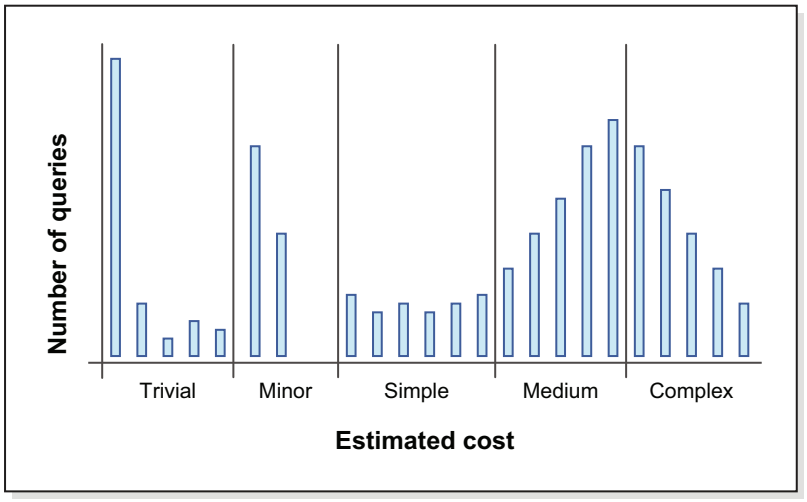


Figure 25. Scenario 2: Modified U-shaped distribution of estimated costs for work

Scenario 3

If the population centers are both of significant size as depicted in Figure 26 on page 95, then it can be beneficial to create a service subclass to separate the populations. When doing this separation, it is necessary to divide the concurrency between the two new service subclasses based on their proportional population by using the technique described in the previous example based on the methodology introduced in "Allocating capacity" on page 30.

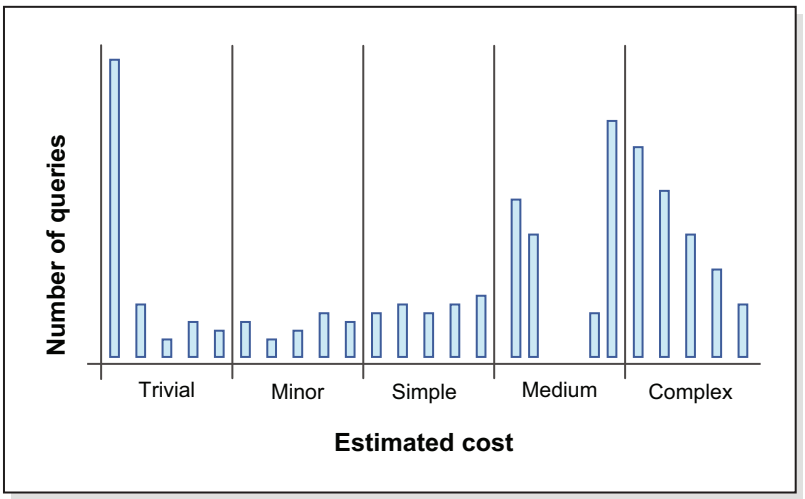


Figure 26. Scenario 3: U-shaped distribution of estimated costs for work

For example, if two large population centers are found in the MEDIUM_DML service subclass, do the following:

1. Adjust the work class set
   a. Modify the upper estimated cost boundary of the MEDIUM_COST_DML work class to be less than the lowest value of the large population center at the upper end of the range
   b. Create the MEDIUM_COMPLEX_COST_DML work class with a lower estimated cost value equal to the value of the previous step, and an upper value equal to the original upper value of the MEDIUM_COST_DML work class range
2. Adjust the DB2 service subclasses as depicted in Figure 27 on page 96
   a. Create the MEDIUM_COMPLEX_DML service subclass
3. Adjust the work action set
   a. Create a MAP_MEDIUM_COST_DML work action for the MEDIUM_COMPLEX_COST_DML work class in the work action set to map any work of that class into the new MEDIUM_COMPLEX_DML service subclass
4. Adjust the DB2 concurrency thresholds
   a. Determine the proportion of the work being moved from the MEDIUM_DML service subclass to the new MEDIUM_COMPLEX_DML service subclass, based on population ratio, and adjust the initial budget allocation to reflect the movement of this amount from the MEDIUM_DML to the MEDIUM_COMPLEX_DML service subclass
      1) Assume 65% movement of the population
      2) Initial budget allocation for MEDIUM_DML service subclass was 10%
      3) Adjusted budget allocation for MEDIUM_DML service subclass is 3.5%
      4) Initial budget allocation for MEDIUM_COMPLEX_DML service subclass is 6.5%
      5) Initial concurrency value for MEDIUM_DML service subclass was 8
      6) Adjusted concurrency threshold value for MEDIUM_DML service subclass is 3
      7) Initial concurrency threshold value for MEDIUM_COMPLEX_DML service subclass is 5

b. Adjust the concurrency threshold on MEDIUM_DML service subclass to the adjusted value (3) for that class
c. Create a concurrency threshold for the new MEDIUM_COMPLEX_DML service subclass with the calculated initial value (5)

5. Create activity thresholds with values appropriate for the population in the MEDIUM_COMPLEX_DML service subclass

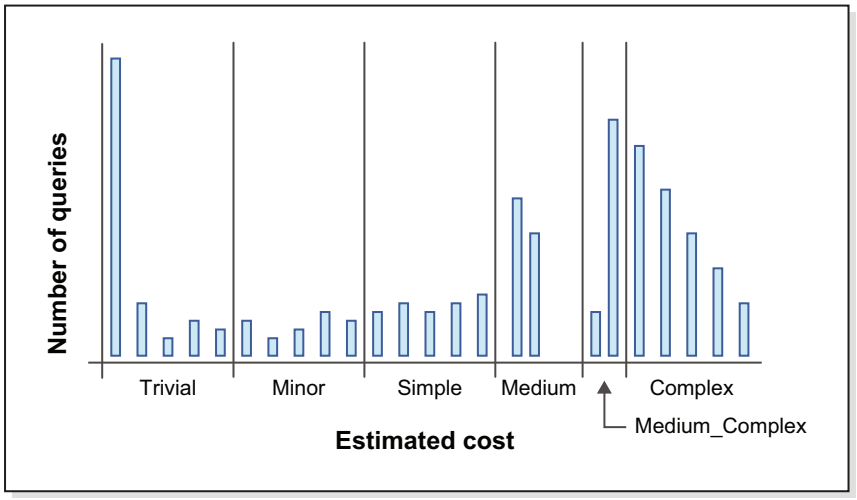


Figure 27. Scenario 3: Modified U-shaped distribution of estimated costs for work

Queries with similar estimated costs

In some cases, you might discover that the estimated cost histogram for a service class has a single bin where the vast majority of statements are tightly packed. This tight packing is referred to as a unimodal spike.

A unimodal spike can happen for the following reasons:

- The range of estimated cost represented by that particular bin encompasses a wider range of values than other points in the histogram and gives the misleading appearance of uniformity to a set of statements in that range.
- The statements being executed are indeed alike in terms of estimated cost.

The primary challenge when facing a unimodal spike scenario is to determine if the set of statements represented by the spike are consistent with a cost-based approach to classifying statements that is the underlying assumption of the methodology in this best practices document.

To do this, we need answers to the following three questions:

1. Is there a relatively constant relationship between the estimated cost and the actual amount of resources consumed by the statements when they are executed?
2. Is the actual resource consumed by the statements of a fundamentally different magnitude than the estimated costs would have led us to expect?
3. Can these statements potentially threaten the stability of the system?

The first question is answered by looking at the range of resource demands made by the work in the target service subclass to see if they are relatively close to each

other. For example, as long as the statements all use roughly the same amount of CPU and I/O when they execute, then the relationship can be said to be constant. However, if you see actual CPU and I/O costs that are an order of magnitude different than each other (for example, 0.1 second of CPU time versus 1.0 second of CPU time), then these statements might be problematic. The important word in the previous sentence is *magnitude*, because we are looking for significant differences in resource consumption.

Statements that use host variables or parameter markers can sometimes display this type of variance in their runtime behavior, despite having the same estimated cost. In cases where the statement uses a host variable or parameter marker, the SQL compiler makes an estimate based on the default value assumptions for data type of the host variable. At execution time, if the actual value supplied is in variance with the compiler assumption, then the resource impact will vary from the estimated cost[8].

Using the captured activity event monitor data, examine the statement text for those statements that would be mapped to the suspect service subclass (that is, the one containing the unimodal spike) based on their estimated cost to see if they are indeed using host variables or parameter markers. If they do, then by capturing activity event monitor data on all database members for these same statements and aggregating this information for each execution, which reflects the overall impact of the statement, you will be able to tell if the resource demands vary widely from one execution to another. For example, if a statement uses host variables or parameter markers, select minimum and maximum values for the **rows_read** and **total_cpu_time** metrics for the same statement text to determine the range of resource consumption for each. If the statements use host variables or parameter markers and have a wide (an order of magnitude) range of impact, then the statements might cause problems depending on which way they vary (answering the second question) and how many of these outliers exist (answering the third question).

The second question is resolved by comparing the actual resource consumed by these problematic statements to the impact originally expected for work in the target service subclass (the one where these statements were mapped based on their estimated cost). Perhaps the best way to do this is to try and find an analog value by comparing the relative size of the actual resource impact (for example, CPU and I/O) of the suspect statements to the resource consumption of statements in the neighboring service subclasses. In other words, we want to be able to answer the question: Is one typical statement in problematic service class Y equivalent to one typical statement from service subclass X, where X is one of the other existing service subclasses (other than Y)?

If the actual CPU and I/O resource impact is equivalent to the resource impact that occurs in a lower cost estimates service subclass than the one to which the problem statements are being mapped, then we are less concerned because the concurrency controls already in place are able to manage this disparity. As an example, this case occurs when statements map to the SIMPLE_DML service subclass based on their cost, but their actual resource impact is equivalent to work running in the MINOR_DML service subclass. The concurrency control on the SIMPLE_DML service subclass is actually more restrictive than if the work had been run in the MINOR_DML service subclass.

8. This variance typically happens when there is a non-uniform distribution of data around the values being provided at run time, where some values have a very large number of associated rows and some have a very small number.

If the actual impact is equivalent to that which occurs in a higher cost service subclass, then we need to be concerned that the concurrency controls are not sufficient. As an example, this case occurs when statements map to the SIMPLE_DML service subclass based on their cost, but their actual resource impact is equivalent to work running in the MEDIUM_DML service subclass. The concurrency control on the SIMPLE_DML service subclass would tend to be less restrictive than if the work had been run in the MEDIUM_DML service subclass which would allow more resources to be consumed than perhaps desired. Again, the degree of concern when this situation arises depends on the answer to how many of these problematic statements exist (the third question).

The answer to the third question essentially boils down to figuring out (roughly) how many of these problematic statements exist at any one time and how large an impact that they have individually. In general, we need to be more concerned about large impact (that is, equivalent to medium or complex SQL) statements all of the time and smaller impact statements only when there is a significant number of them. That is, the system might be able to successfully accommodate an additional, unexpected medium-sized query at that point in time, but not 10 of them.

If statements of concern are found during this investigation and they exist in numbers that can potentially threaten system stability, one of the following actions might remedy the situation:

- Use the **REOPT ALWAYS** options of the **BIND** command to force compilation at run time by using the supplied input values.
- If the statements come from a common source, create a workload and direct this work to a separate service superclass with its own control settings.
- Implement activity thresholds to catch and stop such variants. For more information, see: “Protective measures” on page 39.

Appendix E. SQL for transitioning from stage 1 to stage 2

Sample SQL scripts for transitioning from the stage 1 to the stage 2 best practices configuration.

Sample E1: Sample script for collecting `vmstat` data

The intent of this script is to collect key `vmstat` data from the targeted system and store it for later analysis in conjunction with gathered DB2 monitoring information. Although this data is needed for only the target data module during the transition from a stage 1 to a stage 2 best practices workload management configuration, it can also be run on all member systems in a partitioned database environment and then merged to present an overall perspective about the system.

This script can be run (with minor modifications) by itself or as part of an ongoing `cron` job on any Linux or UNIX system. To have it run every 60 minutes and align with the automatic collection of internal workload management statistics data within the DB2 database manager (controlled by the **wlm_collect_int** database configuration parameter), the `crontab` file contains an entry similar to the following:

```
0,60 * * * * myscript
```

where *myscript* represents the script to be run every 60 minutes.

If you do not want this data being collected forever, modify the above entry to terminate invocation at the time desired or remove it when the need for collection no longer exists. If you want the collection to be ongoing, be sure to implement a script or process to remove unwanted data, after analysis has occurred, to prevent the file size becoming prohibitively large.

Script text:

For an editable version of this script, you can use the `sampleE1.script` file included in the best practices `.zip` file.

```
#!/bin/ksh
#
# This script takes selected columns from vmstat output and inserts
# them into a file for later analysis. The columns extracted below
# will vary depending on the version vmstat you use and the
# environment. On some platforms, the timestamp output is not an
# option for vmstat and the script will have to insert the time from
# another source.
#
# As an example, this script assumes AIX and a non-LPAR environment,
# so the columns used are:
#        $18 timestamp
#        $1  kthr r (Average number of runnable kernel threads)
#        $2  kthr b (Average number of waiting kernel threads)
#        $6  page pi (Pages paged in from paging space)
#        $7  page po (Pages paged out to paging space)
#        $14 cpu us (user time %)
#        $15 cpu sy (system time %)
#        $16 cpu id (idle time %)
#        $17 cpu wa (idle time due I/O wait %)
#
# But if you ran this same command in a shared processor LPAR
# environment, the script would have to be modified to use these columns:
#        $20 timestamp
#        $1  kthr r (Average number of runnable kernel threads)
#        $2  kthr b (Average number of waiting kernel threads)
#        $6  page pi (Pages paged in from paging space)
#        $7  page po (Pages paged out to paging space)
```

```
#        $14 cpu us (user time %)
#        $15 cpu sy (system time %)
#        $16 cpu id (idle time %)
#        $17 cpu wa (idle time due I/O wait %)
#
# If running on another operating system other than AIX non-LPAR, then
# adjust the PARSEDATA line appropriately. Some other common examples
# are provided here:
#
#    Linux
#      PARSEDDATA=`echo $RAWDATA | awk '{print $18,$1,$2,$7,$8,$13,$14,$15,$16}'`
#
#    AIX LPAR
#      PARSEDDATA=`echo $RAWDATA | awk '{print $20,$1,$2,$6,$7,$14,$15,$16,$17}'`
#


#

DATE=`date '+%y%m%d %H:%M:%S`
HOSTNAME=`hostname -s`
FILENAME='vmstat_data.'`echo $HOSTNAME`

# Invoke vmstat with the timestamp option (-t) and ask for 1 second
# interval report. Read last line and parse for columns of interest.

RAWDATA=`vmstat 1 1 | tail -n 1`

# AIX non-LPAR
PARSEDDATA=`echo $RAWDATA | awk '{print $18,$1,$2,$6,$7,$14,$15,$16,$17}'`

# append output to data file
echo $DATE $PARSEDDATA >> $FILENAME

exit 0
```

Example output:

```
111024 15:06:25 95 0 0 0 21 6 73 1
```

Sample E2: Calculating the CPU consumed per service class

This SQL query calculates the delta CPU consumed by each service class on each member and aggregates it over the specified period of time. For the purposes of the use described in the best practices methodology, look at only the sum of the CPU consumption for all of the members on the same physical data module. This data can be obtained by modifying the SQL to consider only those members, or calculating it by hand based on the results of this query.

SQL text:

For an editable version of this SQL script, you can use the `sampleE2.sql` file included in the best practices `.zip` file.

```
--
-- Query to extract and aggregate CPU TIME from statistics event monitor
-- per member on database per day. Include other metrics also used during
-- analysis steps.
--
WITH

-- Determine when each member last started in order for delta metric calculations
  MEMBER_START_TIMES AS
     (SELECT DISTINCT SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP,
             MAX(MESSAGE_TIME) AS FIRSTCONNECT_TIME
      FROM CONTROL_DB2STATISTICS AS CONTROL, SCSTATS_DB2STATISTICS AS SCSTATS
      WHERE MESSAGE = 'FIRST_CONNECT'
         AND CONTROL.PARTITION_NUMBER = SCSTATS.PARTITION_NUMBER
         AND STATISTICS_TIMESTAMP > MESSAGE_TIME
      GROUP BY SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP),

-- Determine how to calculate deltas for each row
  DELTA_METHOD AS
     (SELECT SCSTATS.PARTITION_NUMBER,
             SCSTATS.STATISTICS_TIMESTAMP,
-- If previous row was gathered after last member start, determine delta
```

```
-- by subtraction
             CASE
               WHEN (LAG(SCSTATS.STATISTICS_TIMESTAMP, 1,
                   TIMESTAMP_FORMAT('2007-10-01 23:59:59',
                                    'YYYY-MM-DD HH24:MI:SS'))
                OVER (PARTITION BY SCSTATS.PARTITION_NUMBER
                      ORDER BY SCSTATS.STATISTICS_TIMESTAMP))
                    >=
                    (SELECT FIRSTCONNECT_TIME
                     FROM MEMBER_START_TIMES AS REF
                     WHERE REF.PARTITION_NUMBER = SCSTATS.PARTITION_NUMBER
                        AND REF.STATISTICS_TIMESTAMP
                                          = SCSTATS.STATISTICS_TIMESTAMP)

               THEN 'Y'
               ELSE 'N'
             END AS SUBTRACT_FROM_PREV_ROW
      FROM SCSTATS_DB2STATISTICS AS SCSTATS
      GROUP BY SCSTATS.PARTITION_NUMBER, SCSTATS.STATISTICS_TIMESTAMP
      ORDER BY SCSTATS.PARTITION_NUMBER, SCSTATS.STATISTICS_TIMESTAMP),


-- extract wanted metrics from DETAILS_XML column in service class statistics
  V1 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             METRIC_NAME, VALUE
      FROM SCSTATS_DB2STATISTICS,
           TABLE (MON_FORMAT_XML_METRICS_BY_ROW(DETAILS_XML))
      WHERE METRIC_NAME IN ('TOTAL_CPU_TIME', 'TOTAL_WAIT_TIME')),


-- Pivot data into table format
  V2 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             MAX(DECODE(METRIC_NAME, 'TOTAL_CPU_TIME', VALUE))
                                                  AS TOTAL_CPU_TIME,
             MAX(DECODE(METRIC_NAME, 'TOTAL_WAIT_TIME', VALUE))
                                                  AS TOTAL_WAIT_TIME
      FROM V1
      GROUP BY STATISTICS_TIMESTAMP, PARTITION_NUMBER,
               SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME),


-- Calculate delta values for the extracted values based on delta calculation
-- method
  V3 AS
     (SELECT V2.STATISTICS_TIMESTAMP, V2.PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_CPU_TIME - LAG(TOTAL_CPU_TIME, 1, 0)
                                   OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                      SERVICE_SUPERCLASS_NAME,
                                                      SERVICE_SUBCLASS_NAME
                                         ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE TOTAL_CPU_TIME
             END AS TOTAL_CPU_TIME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_WAIT_TIME - LAG(TOTAL_WAIT_TIME, 1, 0)
                                    OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                       SERVICE_SUPERCLASS_NAME,
                                                       SERVICE_SUBCLASS_NAME
                                          ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE TOTAL_WAIT_TIME
             END AS TOTAL_WAIT_TIME
    FROM V2, DELTA_METHOD
    WHERE V2.PARTITION_NUMBER = DELTA_METHOD.PARTITION_NUMBER
       AND V2.STATISTICS_TIMESTAMP = DELTA_METHOD.STATISTICS_TIMESTAMP),

-- Aggregate extracted metrics to get total consumption per service class
-- per member per day
V4 AS
   (SELECT DATE(SCSTATS.STATISTICS_TIMESTAMP) AS INTERVAL_DATE,
```

```
        SCSTATS.PARTITION_NUMBER,
        SUBSTR(SCSTATS.SERVICE_SUPERCLASS_NAME,1,32)
                                              AS SERVICE_SUPERCLASS_NAME,
        SUBSTR(SCSTATS.SERVICE_SUBCLASS_NAME,1,32) AS SERVICE_SUBCLASS_NAME,
        SUM(V3.TOTAL_CPU_TIME) AS TOTAL_CPU_TIME,
        SUM(V3.TOTAL_WAIT_TIME) AS TOTAL_WAIT_TIME,
        MAX(SCSTATS.COORD_ACT_LIFETIME_AVG) AS LIFETIME_AVG,
        MAX(SCSTATS.COORD_ACT_LIFETIME_TOP) AS LIFETIME_MAX
    FROM SCSTATS_DB2STATISTICS AS SCSTATS, V3
    WHERE SCSTATS.STATISTICS_TIMESTAMP = V3.STATISTICS_TIMESTAMP
      AND SCSTATS.SERVICE_SUPERCLASS_NAME = V3.SERVICE_SUPERCLASS_NAME
      AND SCSTATS.SERVICE_SUBCLASS_NAME = V3.SERVICE_SUBCLASS_NAME
      AND SCSTATS.PARTITION_NUMBER = V3.PARTITION_NUMBER
    GROUP BY DATE(SCSTATS.STATISTICS_TIMESTAMP),
             SCSTATS.PARTITION_NUMBER,
             SCSTATS.SERVICE_SUPERCLASS_NAME,
             SCSTATS.SERVICE_SUBCLASS_NAME)

-- Report final results
SELECT *
FROM V4
ORDER BY INTERVAL_DATE,
         PARTITION_NUMBER,
         SERVICE_SUPERCLASS_NAME,
         SERVICE_SUBCLASS_NAME;
```

Example output:

```
INTERVAL_DATE PARTITION_NUMBER SERVICE_SUPERCLASS_NAME          SERVICE_SUBCLASS_NAME
------------- ---------------- ------------------------------- --------------------
10/20/2011                 10 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/20/2011                 10 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/20/2011                 10 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/20/2011                 10 WLMBP_MASTER                    COMPLEX_DML
10/20/2011                 10 WLMBP_MASTER                    ETL
10/20/2011                 10 WLMBP_MASTER                    MEDIUM_DML
10/20/2011                 10 WLMBP_MASTER                    MINOR_DML
10/20/2011                 10 WLMBP_MASTER                    SIMPLE_DML
10/20/2011                 10 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/20/2011                 10 WLMBP_MASTER                    TRIVIAL_DML
10/20/2011                 20 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/20/2011                 20 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/20/2011                 20 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/20/2011                 20 WLMBP_MASTER                    COMPLEX_DML
10/20/2011                 20 WLMBP_MASTER                    ETL
10/20/2011                 20 WLMBP_MASTER                    MEDIUM_DML
10/20/2011                 20 WLMBP_MASTER                    MINOR_DML
10/20/2011                 20 WLMBP_MASTER                    SIMPLE_DML
10/20/2011                 20 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/20/2011                 20 WLMBP_MASTER                    TRIVIAL_DML
10/20/2011                 30 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/20/2011                 30 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/20/2011                 30 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/20/2011                 30 WLMBP_MASTER                    COMPLEX_DML
10/20/2011                 30 WLMBP_MASTER                    ETL
10/20/2011                 30 WLMBP_MASTER                    MEDIUM_DML
10/20/2011                 30 WLMBP_MASTER                    MINOR_DML
10/20/2011                 30 WLMBP_MASTER                    SIMPLE_DML
10/20/2011                 30 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/20/2011                 30 WLMBP_MASTER                    TRIVIAL_DML
10/20/2011                 40 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/20/2011                 40 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/20/2011                 40 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/20/2011                 40 WLMBP_MASTER                    COMPLEX_DML
10/20/2011                 40 WLMBP_MASTER                    ETL
10/20/2011                 40 WLMBP_MASTER                    MEDIUM_DML
10/20/2011                 40 WLMBP_MASTER                    MINOR_DML
10/20/2011                 40 WLMBP_MASTER                    SIMPLE_DML
10/20/2011                 40 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/20/2011                 40 WLMBP_MASTER                    TRIVIAL_DML
10/21/2011                 10 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/21/2011                 10 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/21/2011                 10 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/21/2011                 10 WLMBP_MASTER                    COMPLEX_DML
10/21/2011                 10 WLMBP_MASTER                    ETL
10/21/2011                 10 WLMBP_MASTER                    MEDIUM_DML
10/21/2011                 10 WLMBP_MASTER                    MINOR_DML
10/21/2011                 10 WLMBP_MASTER                    SIMPLE_DML
10/21/2011                 10 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
```

```
10/21/2011              10 WLMBP_MASTER                 TRIVIAL_DML
10/21/2011              20 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/21/2011              20 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/21/2011              20 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/21/2011              20 WLMBP_MASTER                 COMPLEX_DML
10/21/2011              20 WLMBP_MASTER                 ETL
10/21/2011              20 WLMBP_MASTER                 MEDIUM_DML
10/21/2011              20 WLMBP_MASTER                 MINOR_DML
10/21/2011              20 WLMBP_MASTER                 SIMPLE_DML
10/21/2011              20 WLMBP_MASTER                 SYSDEFAULTSUBCLASS
10/21/2011              20 WLMBP_MASTER                 TRIVIAL_DML
10/21/2011              30 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/21/2011              30 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/21/2011              30 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/21/2011              30 WLMBP_MASTER                 COMPLEX_DML
10/21/2011              30 WLMBP_MASTER                 ETL
10/21/2011              30 WLMBP_MASTER                 MEDIUM_DML
10/21/2011              30 WLMBP_MASTER                 MINOR_DML
10/21/2011              30 WLMBP_MASTER                 SIMPLE_DML
10/21/2011              30 WLMBP_MASTER                 SYSDEFAULTSUBCLASS
10/21/2011              30 WLMBP_MASTER                 TRIVIAL_DML
10/21/2011              40 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/21/2011              40 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/21/2011              40 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/21/2011              40 WLMBP_MASTER                 COMPLEX_DML
10/21/2011              40 WLMBP_MASTER                 ETL
10/21/2011              40 WLMBP_MASTER                 MEDIUM_DML
10/21/2011              40 WLMBP_MASTER                 MINOR_DML
10/21/2011              40 WLMBP_MASTER                 SIMPLE_DML
10/21/2011              40 WLMBP_MASTER                 SYSDEFAULTSUBCLASS
10/21/2011              40 WLMBP_MASTER                 TRIVIAL_DML
10/22/2011              10 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/22/2011              10 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/22/2011              10 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/22/2011              10 WLMBP_MASTER                 COMPLEX_DML
10/22/2011              10 WLMBP_MASTER                 ETL
10/22/2011              10 WLMBP_MASTER                 MEDIUM_DML
10/22/2011              10 WLMBP_MASTER                 MINOR_DML
10/22/2011              10 WLMBP_MASTER                 SIMPLE_DML
10/22/2011              10 WLMBP_MASTER                 SYSDEFAULTSUBCLASS
10/22/2011              10 WLMBP_MASTER                 TRIVIAL_DML
10/22/2011              20 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/22/2011              20 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/22/2011              20 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/22/2011              20 WLMBP_MASTER                 COMPLEX_DML
10/22/2011              20 WLMBP_MASTER                 ETL
10/22/2011              20 WLMBP_MASTER                 MEDIUM_DML
10/22/2011              20 WLMBP_MASTER                 MINOR_DML
10/22/2011              20 WLMBP_MASTER                 SIMPLE_DML
10/22/2011              20 WLMBP_MASTER                 SYSDEFAULTSUBCLASS
10/22/2011              20 WLMBP_MASTER                 TRIVIAL_DML
10/22/2011              30 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/22/2011              30 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/22/2011              30 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/22/2011              30 WLMBP_MASTER                 COMPLEX_DML
10/22/2011              30 WLMBP_MASTER                 ETL
10/22/2011              30 WLMBP_MASTER                 MEDIUM_DML
10/22/2011              30 WLMBP_MASTER                 MINOR_DML
10/22/2011              30 WLMBP_MASTER                 SIMPLE_DML
10/22/2011              30 WLMBP_MASTER                 SYSDEFAULTSUBCLASS
10/22/2011              30 WLMBP_MASTER                 TRIVIAL_DML
10/22/2011              40 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/22/2011              40 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/22/2011              40 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/22/2011              40 WLMBP_MASTER                 COMPLEX_DML
10/22/2011              40 WLMBP_MASTER                 ETL
10/22/2011              40 WLMBP_MASTER                 MEDIUM_DML
10/22/2011              40 WLMBP_MASTER                 MINOR_DML
10/22/2011              40 WLMBP_MASTER                 SIMPLE_DML
10/22/2011              40 WLMBP_MASTER                 SYSDEFAULTSUBCLASS
10/22/2011              40 WLMBP_MASTER                 TRIVIAL_DML
10/23/2011              10 SYSDEFAULTMAINTENANCECLASS   SYSDEFAULTSUBCLASS
10/23/2011              10 SYSDEFAULTSYSTEMCLASS        SYSDEFAULTSUBCLASS
10/23/2011              10 SYSDEFAULTUSERCLASS          SYSDEFAULTSUBCLASS
10/23/2011              10 WLMBP_MASTER                 COMPLEX_DML
10/23/2011              10 WLMBP_MASTER                 ETL
10/23/2011              10 WLMBP_MASTER                 MEDIUM_DML
10/23/2011              10 WLMBP_MASTER                 MINOR_DML
10/23/2011              10 WLMBP_MASTER                 SIMPLE_DML
```

```
10/23/2011               10 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/23/2011               10 WLMBP_MASTER                    TRIVIAL_DML
10/23/2011               20 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/23/2011               20 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/23/2011               20 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/23/2011               20 WLMBP_MASTER                    COMPLEX_DML
10/23/2011               20 WLMBP_MASTER                    ETL
10/23/2011               20 WLMBP_MASTER                    MEDIUM_DML
10/23/2011               20 WLMBP_MASTER                    MINOR_DML
10/23/2011               20 WLMBP_MASTER                    SIMPLE_DML
10/23/2011               20 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/23/2011               20 WLMBP_MASTER                    TRIVIAL_DML
10/23/2011               30 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/23/2011               30 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/23/2011               30 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/23/2011               30 WLMBP_MASTER                    COMPLEX_DML
10/23/2011               30 WLMBP_MASTER                    ETL
10/23/2011               30 WLMBP_MASTER                    MEDIUM_DML
10/23/2011               30 WLMBP_MASTER                    MINOR_DML
10/23/2011               30 WLMBP_MASTER                    SIMPLE_DML
10/23/2011               30 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/23/2011               30 WLMBP_MASTER                    TRIVIAL_DML
10/23/2011               40 SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS
10/23/2011               40 SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS
10/23/2011               40 SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS
10/23/2011               40 WLMBP_MASTER                    COMPLEX_DML
10/23/2011               40 WLMBP_MASTER                    ETL
10/23/2011               40 WLMBP_MASTER                    MEDIUM_DML
10/23/2011               40 WLMBP_MASTER                    MINOR_DML
10/23/2011               40 WLMBP_MASTER                    SIMPLE_DML
10/23/2011               40 WLMBP_MASTER                    SYSDEFAULTSUBCLASS
10/23/2011               40 WLMBP_MASTER                    TRIVIAL_DML
```

TOTAL_CPU_TIME	TOTAL_WAIT_TIME	LIFETIME_AVG	LIFETIME_MAX
106704777	6540669	-1	-1
287693	2026	-1	-1
4847566	116184	-1	-1
17254	90431	2208	5823
0	0	0	0
383977	471556	24711	279884
3957955	9004764	15877	405912
692286	3679570	81197	284552
5887718297	214174636	-1	-1
519176059	914588220	578	700813
7129145	302553	-1	-1
353750	13659	-1	-1
1271981	19799	-1	-1
0	48672	0	0
0	0	0	0
4589	42850	0	0
312137	3439399	0	0
163820	2047096	0	0
747306013	45689523	-1	-1
3381723	310035711	0	0
7138197	293841	-1	-1
0	0	-1	-1
1370473	20709	-1	-1
0	53965	0	0
0	0	0	0
9592	38413	0	0
1256513	4124180	0	0
120012	2106774	0	0
763389488	46899223	-1	-1
3215259	320188161	0	0
7207650	301957	-1	-1
0	0	-1	-1
3421679	19466	-1	-1
0	32289	0	0
0	0	0	0
841734	336566	0	0
140354	4068548	0	0
454107	2182522	0	0
771141497	48673873	-1	-1
5702104	332305497	0	0
203145321	11582957	-1	-1
594724	0	-1	-1
0	0	-1	-1

195524	688157	12792	35102
0	0	0	0
295393	1775624	175382	216402
74425709	86458716	22263	1032808
24607626	6821937	200997	253347
14882935716	523964689	-1	-1
2049210624	4098016790	583	1902433
7657356	1749450	-1	-1
32133	6148	-1	-1
0	0	-1	-1
0	296411	0	0
0	0	0	0
1990160	869638	0	0
17117670	31032065	0	0
2290176	2811091	0	0
5881246549	122267483	-1	-1
64473686	1567362052	0	0
7724815	1477124	-1	-1
0	0	-1	-1
0	0	-1	-1
0	302117	0	0
0	0	0	0
522130	678387	0	0
15143195	29309475	0	0
206547	2634596	0	0
5869541349	86109132	-1	-1
52627692	1563555146	0	0
7685600	1405801	-1	-1
0	0	-1	-1
0	0	-1	-1
2307	334794	0	0
0	0	0	0
8662531	1013182	0	0
64400780	35289653	0	0
3928648	2589308	0	0
5916400767	93862152	-1	-1
66111089	1503418504	0	0
289358897	12238238	-1	-1
821277	0	-1	-1
0	0	-1	-1
346556	780827	7488	37322
0	0	0	0
418519	4394890	256698	510703
159820231	163739501	35854	1604595
285164682	57251917	1187717	1187717
20107078363	636797136	-1	-1
3305198279	5213304935	756	2815836
5215671	363795	-1	-1
0	0	-1	-1
0	0	-1	-1
27507	348636	0	0
0	0	0	0
33589145	2961423	0	0
138876245	52637926	0	0
21819191	14828942	0	0
10690633282	213714550	-1	-1
534089481	2001975980	0	0
5248892	371974	-1	-1
0	0	-1	-1
0	0	-1	-1
10842	366035	0	0
0	0	0	0
261086	1806819	0	0
115234395	52783373	0	0
6818931	11565758	0	0
10651771861	139805401	-1	-1
467447171	1957975092	0	0
5238245	351706	-1	-1
0	0	-1	-1
0	0	-1	-1
0	353719	0	0
0	0	0	0
121943	1689048	0	0
105132934	44401368	0	0
5958238	11214203	0	0
10421611163	116122315	-1	-1
309020504	1670049841	0	0
25537305	1719464	-1	-1
85210	0	-1	-1

```
                   0                   0                  -1                  -1
               28111               85654                1511                2779
                   0                   0                   0                   0
                   0                   0                   0                   0
            15642733            34551345               52741              813009
            20625470             8717787              182720              735245
          2505357225            65263905                  -1                  -1
           407796627           525696489                 414             2097374
              549625               18036                  -1                  -1
                   0                   0                  -1                  -1
                   0                   0                  -1                  -1
                   0               49331                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
            28275381            13773358                   0                   0
             2901046             1324477                   0                   0
          1390676358            25235079                  -1                  -1
           118305406           204464021                   0                   0
              536013               17525                  -1                  -1
                   0                   0                  -1                  -1
                   0                   0                  -1                  -1
                   0               56800                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
            32238350            14952813                   0                   0
            10808186             2473607                   0                   0
          1366045132            13298611                  -1                  -1
            30006553           192317610                   0                   0
              537782               19828                  -1                  -1
                   0                   0                  -1                  -1
                   0                   0                  -1                  -1
                   0               50851                   0                   0
                   0                   0                   0                   0
                   0                   0                   0                   0
            15518907            12811813                   0                   0
            13188937             2411070                   0                   0
          1398755710            10947642                  -1                  -1
            51630861           162752964                   0                   0

  160 record(s) selected.
```

Sample E3: Determining the maximum activity lifetime value from statistics event monitor data

This SQL query calculates the maximum lifetime observed for an activity at any service class from the data available in the statistics event monitor.

SQL text:

For an editable version of this SQL script, you can use the `sampleE3.sql` file included in the best practices `.zip` file.

```
--
-- Query to determine the longest activity time seen in each service class
--
SELECT DATE(STATISTICS_TIMESTAMP) AS INTERVAL_DATE,
       SUBSTR(SERVICE_SUPERCLASS_NAME,1,32) AS SERVICE_SUPERCLASS_NAME,
       SUBSTR(SERVICE_SUBCLASS_NAME,1,32) AS SERVICE_SUBCLASS_NAME,
       MAX(COORD_ACT_LIFETIME_AVG) AS LIFETIME_AVG,
       MAX(COORD_ACT_LIFETIME_TOP) AS LIFETIME_MAX
FROM SCSTATS_DB2STATISTICS
GROUP BY DATE(STATISTICS_TIMESTAMP), SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME
ORDER BY DATE(STATISTICS_TIMESTAMP), SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME;
```

Example output:

```
INTERVAL_DATE SERVICE_SUPERCLASS_NAME          SERVICE_SUBCLASS_NAME            LIFETIME_AVG         LIFETIME_MAX
------------- ------------------------------- ------------------------------- -------------------- --------------------
10/20/2011    SYSDEFAULTMAINTENANCECLASS      SYSDEFAULTSUBCLASS                             -1                   -1
10/20/2011    SYSDEFAULTSYSTEMCLASS           SYSDEFAULTSUBCLASS                             -1                   -1
10/20/2011    SYSDEFAULTUSERCLASS             SYSDEFAULTSUBCLASS                             -1                   -1
10/20/2011    WLMBP_MASTER                    COMPLEX_DML                                  2208                 5823
10/20/2011    WLMBP_MASTER                    ETL                                             0                    0
10/20/2011    WLMBP_MASTER                    MEDIUM_DML                                  24711               279884
10/20/2011    WLMBP_MASTER                    MINOR_DML                                   15877               405912
10/20/2011    WLMBP_MASTER                    SIMPLE_DML                                  81197               284552
10/20/2011    WLMBP_MASTER                    SYSDEFAULTSUBCLASS                             -1                   -1
```

```
10/20/2011    WLMBP_MASTER                  TRIVIAL_DML                          578              700813
10/21/2011    SYSDEFAULTMAINTENANCECLASS    SYSDEFAULTSUBCLASS                    -1                  -1
10/21/2011    SYSDEFAULTSYSTEMCLASS         SYSDEFAULTSUBCLASS                    -1                  -1
10/21/2011    SYSDEFAULTUSERCLASS           SYSDEFAULTSUBCLASS                    -1                  -1
10/21/2011    WLMBP_MASTER                  COMPLEX_DML                        12792               35102
10/21/2011    WLMBP_MASTER                  ETL                                    0                   0
10/21/2011    WLMBP_MASTER                  MEDIUM_DML                        175382              216402
10/21/2011    WLMBP_MASTER                  MINOR_DML                          22263             1032808
10/21/2011    WLMBP_MASTER                  SIMPLE_DML                        200997              253347
10/21/2011    WLMBP_MASTER                  SYSDEFAULTSUBCLASS                    -1                  -1
10/21/2011    WLMBP_MASTER                  TRIVIAL_DML                          583             1902433
10/22/2011    SYSDEFAULTMAINTENANCECLASS    SYSDEFAULTSUBCLASS                    -1                  -1
10/22/2011    SYSDEFAULTSYSTEMCLASS         SYSDEFAULTSUBCLASS                    -1                  -1
10/22/2011    SYSDEFAULTUSERCLASS           SYSDEFAULTSUBCLASS                    -1                  -1
10/22/2011    WLMBP_MASTER                  COMPLEX_DML                         7488               37322
10/22/2011    WLMBP_MASTER                  ETL                                    0                   0
10/22/2011    WLMBP_MASTER                  MEDIUM_DML                        256698              510703
10/22/2011    WLMBP_MASTER                  MINOR_DML                          35854             1604595
10/22/2011    WLMBP_MASTER                  SIMPLE_DML                       1187717             1187717
10/22/2011    WLMBP_MASTER                  SYSDEFAULTSUBCLASS                    -1                  -1
10/22/2011    WLMBP_MASTER                  TRIVIAL_DML                          756             2815836
10/23/2011    SYSDEFAULTMAINTENANCECLASS    SYSDEFAULTSUBCLASS                    -1                  -1
10/23/2011    SYSDEFAULTSYSTEMCLASS         SYSDEFAULTSUBCLASS                    -1                  -1
10/23/2011    SYSDEFAULTUSERCLASS           SYSDEFAULTSUBCLASS                    -1                  -1
10/23/2011    WLMBP_MASTER                  COMPLEX_DML                         1511                2779
10/23/2011    WLMBP_MASTER                  ETL                                    0                   0
10/23/2011    WLMBP_MASTER                  MEDIUM_DML                             0                   0
10/23/2011    WLMBP_MASTER                  MINOR_DML                          52741              813009
10/23/2011    WLMBP_MASTER                  SIMPLE_DML                        182720              735245
10/23/2011    WLMBP_MASTER                  SYSDEFAULTSUBCLASS                    -1                  -1
10/23/2011    WLMBP_MASTER                  TRIVIAL_DML                          414             2097374

  40 record(s) selected.
```

Sample E4: Viewing connection attributes and DB2 workload assignments from activity event monitor data

This SQL query shows the set of critical connection attributes for each activity, as well as the DB2 workload definition to which it was assigned.

SQL text:

For an editable version of this SQL script, you can use the `sampleE4.sql` file included in the best practices `.zip` file.

```
--
-- This query shows the connection attributes of all transactions that mapped
-- to each workload definition based on the information in the activity event
-- monitor
--
SELECT DISTINCT COORD_PARTITION_NUM AS ADMIN_NODE,
       WORKLOAD_ID, WORKLOADNAME,
       ADDRESS,
       APPL_NAME AS APPLNAME,
       SESSION_AUTH_ID AS SESSION_USER,
       TPMON_ACC_STR AS CLIENT_ACCTNG,
       TPMON_CLIENT_USERID AS CLIENT_USERID,
       TPMON_CLIENT_APP AS CLIENT_APPLNAME,
       TPMON_CLIENT_WKSTN AS CLIENT_WRKSTNNAME
FROM ACTIVITY_DB2ACTIVITIES, SYSCAT.WORKLOADS AS REFERENCE
WHERE WORKLOAD_ID = REFERENCE.WORKLOADID
  AND PARTITION_NUMBER = COORD_PARTITION_NUM
ORDER BY WORKLOAD_ID, COORD_PARTITION_NUM;
```

Example output:

```
ADMIN_NODE  WORKLOAD_ID WORKLOADNAME            ADDRESS          APPLNAME
----------- ----------- ----------------------- ---------------- --------------------
         10           1 SYSDEFAULTUSERWORKLOAD                   deletes
         10           1 SYSDEFAULTUSERWORKLOAD                   fltables
         10           1 SYSDEFAULTUSERWORKLOAD                   selects
         10           1 SYSDEFAULTUSERWORKLOAD                   supdates
         10           1 SYSDEFAULTUSERWORKLOAD                   uptables
         10           1 SYSDEFAULTUSERWORKLOAD                   db2bp
         10           1 SYSDEFAULTUSERWORKLOAD                   db2bp
         10           1 SYSDEFAULTUSERWORKLOAD                   db2bp
         10           1 SYSDEFAULTUSERWORKLOAD  300.300.300.300  db2jcc_application
```

```
SESSION_USER  CLIENT_ACCTNG  CLIENT_USERID  CLIENT_APPLNAME     CLIENT_WRKSTNNAME
------------ -------------- -------------- ------------------- ------------------
SVTPDB2
SVTPDB2
SVTPDB2
SVTPDB2
SVTPDB2
SVTPDB2                                    CLP sampleF1.sql
SVTPDB2                                    CLP sampleF2.sql
SVTPDB2                                    CLP sampleF4.sql
SVTPDB2                                                        henry

  9 record(s) selected.
```

Appendix F. SQL for maintaining a stable stage 2 configuration

Sample F1: Summary report on threshold violations

This SQL query presents a simple summary report on the threshold violation data present in the thresholds event monitor.

SQL text:

For an editable version of this SQL script, you can use the `sampleF1.sql` file included in the best practices `.zip` file.

```
--
-- Query to present a summary of all threshold violations present
-- in the threshold violations event monitor data
--
SELECT TIME_OF_VIOLATION,
-- if threshold is still defined in catalogs, report its name
       COALESCE(THRESHOLDNAME, 'UNKNOWN(' || VIOL.THRESHOLDID || ')') AS THRESHOLD_NAME,
       VIOL.THRESHOLD_ACTION,
       VIOL.THRESHOLD_MAXVALUE AS MAX_VALUE,
       VIOL.ACTIVITY_COLLECTED AS COLLECTED,
       VIOL.APPL_ID,
       VIOL.UOW_ID,
       VIOL.ACTIVITY_ID,
-- If the activity data is present, provide user authorization ID
       CASE
         WHEN (VIOL.ACTIVITY_COLLECTED = 'Y')
         THEN COALESCE(SESSION_AUTH_ID, 'NOT AVAILABLE')
         ELSE NULL
       END AS SESSION_USER,
-- If the activity data is present, report the first 100 bytes of text
       CASE
         WHEN (ACTIVITY_COLLECTED = 'Y')
         THEN COALESCE(SUBSTR(STMT_TEXT, 1, 100), 'NOT AVAILABLE')
         ELSE NULL
       END AS TEXT_FRAGMENT
FROM THRESHOLDVIOLATIONS_DB2THRESHOLDS AS VIOL, SYSCAT.THRESHOLDS AS CAT,
     ACTIVITY_DB2ACTIVITIES AS ACT, ACTIVITYSTMT_DB2ACTIVITIES AS DATA
WHERE VIOL.THRESHOLDID = CAT.THRESHOLDID
  AND VIOL.APPL_ID = ACT.APPL_ID
  AND VIOL.UOW_ID = ACT.UOW_ID
  AND VIOL.ACTIVITY_ID = ACT.ACTIVITY_ID
  AND VIOL.APPL_ID = DATA.APPL_ID
  AND VIOL.UOW_ID = DATA.UOW_ID
  AND VIOL.ACTIVITY_ID = DATA.ACTIVITY_ID
ORDER BY TIME_OF_VIOLATION;
```

Example output:

```
TIME_OF_VIOLATION          THRESHOLD_NAME              THRESHOLD_ACTION MAX_VALUE
-------------------------- --------------------------- ---------------- --------------------
2011-10-20-12.10.35.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.10.52.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.10.57.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                           60
2011-10-20-12.11.18.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.12.08.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.12.53.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.14.43.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.17.04.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                           60
2011-10-20-12.18.17.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.19.10.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.19.51.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                           60
2011-10-20-12.20.12.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.20.57.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.23.58.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.24.37.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.24.48.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                           60
2011-10-20-12.24.53.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
2011-10-20-12.29.01.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
.
.
.
2011-10-23-02.03.53.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                           60
2011-10-23-02.04.16.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                           60
```

```
2011-10-23-02.04.26.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                          60
2011-10-23-02.04.41.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.06.36.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.06.42.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                          60
2011-10-23-02.07.56.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.07.56.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.08.23.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                          60
2011-10-23-02.08.40.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.08.55.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                          60
2011-10-23-02.10.17.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.12.15.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.12.25.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.12.29.000000 WLMBP_TRIVIAL_DML_TIMEOUT   Continue                          60
2011-10-23-02.12.32.000000 WLMBP_MINOR_DML_TIMEOUT     Continue                          60
```

```
COLLECTED APPL_ID                      UOW_ID      ACTIVITY_ID          SESSION_USER
--------- ---------------------------- ----------- -------------------- --------------
Y         *N10.svtpdb2.111020161448            6                    1 SVTPDB2
Y         *N10.svtpdb2.111020161556            1                   44 SVTPDB2
Y         *N10.svtpdb2.111020161513            3                    1 SVTPDB2
Y         *N10.svtpdb2.111020161507            7                    1 SVTPDB2
Y         *N10.svtpdb2.111020161617            3                    1 SVTPDB2
Y         *N10.svtpdb2.111020161709            1                   45 SVTPDB2
Y         *N10.svtpdb2.111020161810            2                    1 SVTPDB2
Y         *N10.svtpdb2.111020161938            4                    1 SVTPDB2
Y         *N10.svtpdb2.111020162058            1                   38 SVTPDB2
Y         *N10.svtpdb2.111020161856           11                    1 SVTPDB2
Y         *N10.svtpdb2.111020162201            4                    1 SVTPDB2
Y         *N10.svtpdb2.111020162220            2                    1 SVTPDB2
Y         *N10.svtpdb2.111020162309            1                   40 SVTPDB2
Y         *N10.svtpdb2.111020162535            1                   49 SVTPDB2
Y         *N10.svtpdb2.111020162547            6                    1 SVTPDB2
Y         *N10.svtpdb2.111020162549            2                    1 SVTPDB2
Y         *N10.svtpdb2.111020162605            3                    1 SVTPDB2
Y         *N10.svtpdb2.111020162933            1                   32 SVTPDB2
.
.
.
Y         *N10.svtpdb2.111101135049            1                   45 SVTPDB2
Y         *N10.svtpdb2.111101134955            1                   26 SVTPDB2
Y         *N10.svtpdb2.111101135634            2                    1 SVTPDB2
Y         *N10.svtpdb2.111101135422            1                   17 SVTPDB2
Y         *N10.svtpdb2.111101101519           11                    1 SVTPDB2
Y         *N10.svtpdb2.111101135052            6                    1 SVTPDB2
Y         *N10.svtpdb2.111101101519           12                    1 SVTPDB2
Y         *N10.svtpdb2.111101134955            4                    1 SVTPDB2
Y         *N10.svtpdb2.111101143128           11                    1 SVTPDB2
Y         *N10.svtpdb2.111101131530            3                    1 SVTPDB2
Y         *N10.svtpdb2.111101135052           16                    1 SVTPDB2
Y         *N10.svtpdb2.111101144709            1                   45 SVTPDB2
Y         *N10.svtpdb2.111101145051           10                    1 SVTPDB2
Y         *N10.svtpdb2.111101134955            8                    1 SVTPDB2
Y         *N10.svtpdb2.111101145350            3                    1 SVTPDB2
Y         *N10.svtpdb2.111101145601            1                   45 SVTPDB2
```

```
TEXT_FRAGMENT
-----------------------------------------------------------------------------------------------
SELECT 1 FROM "T3LVL2"."TABLE91" FOR UPDATE OF "Col1492_DUP", "Col1491_DUP", "Col1483_UNQ", "Col1494
SELECT 1 FROM "T3LVL2"."TABLE61" FOR UPDATE OF "Col1023_DUP", "Col1033_UNQ", "Col1031_DUP", "Col1030
SELECT 1 FROM "T3LVL2"."TABLE3" FOR UPDATE OF "Col20_DUP", "Col21_UNQ", "Col22_UNQ", "Col23_UNQ", "C
SELECT 1 FROM "T3LVL2"."TABLE59" FOR UPDATE OF "Col1000_UNQ", "Col1010_UNQ", "Col1004_UNQ", "Col1006
SELECT 1 FROM "T3LVL2"."TABLE48" FOR UPDATE OF "Col777_DUP", "Col789_DUP", "Col805_UNQ", "Col784_DUP
SELECT 1 FROM "T3LVL2"."TABLE90" FOR UPDATE OF "Col1452_DUP", "Col1455_DUP", "Col1458_DUP", "Col1469
SELECT 1 FROM "T3LVL2"."TABLE65" FOR UPDATE OF "Col1080_UNQ", "Col1077_UNQ", "Col1083_UNQ", "Col1087
SELECT 1 FROM "T3LVL2"."TABLE13" FOR UPDATE OF "Col221_DUP", "Col211_UNQ", "Col202_DUP", "Col216_UNQ
SELECT DISTINCT "Col501_DUP" FROM "T3LVL2"."TABLE29" FETCH FIRST 10 ROWS ONLY
SELECT 1 FROM "T3LVL2"."TABLE60" FOR UPDATE OF "Col1014_DUP", "Col1013_DUP"
SELECT 1 FROM "T3LVL2"."TABLE3" FOR UPDATE OF "Col22_UNQ", "Col19_UNQ", "Col23_UNQ", "Col21_UNQ", "C
SELECT 1 FROM "T3LVL2"."TABLE56" FOR UPDATE OF "Col986_DUP", "Col973_UNQ", "Col976_DUP", "Col989_DUP
SELECT 1 FROM "T3LVL2"."TABLE87" FOR UPDATE OF "Col1383_UNQ", "Col1386_DUP", "Col1393_UNQ", "Col1382
SELECT COUNT(*) FROM "T3LVL2"."TABLE92" FOR READ ONLY WITH RR
SELECT 1 FROM "T3LVL2"."TABLE56" FOR UPDATE OF "Col982_DUP", "Col987_UNQ", "Col988_DUP", "Col979_DUP
SELECT 1 FROM "T3LVL2"."TABLE12" FOR UPDATE OF "Col181_UNQ", "Col187_UNQ", "Col185_UNQ", "Col194_UNQ
SELECT 1 FROM "T3LVL2"."TABLE96" FOR UPDATE OF "Col1577_DUP", "Col1576_DUP", "Col1571_DUP", "Col1586
SELECT DISTINCT "Col1125_DUP" FROM "T3LVL2"."TABLE70" FETCH FIRST 10 ROWS ONLY
.
.
.
SELECT COUNT(*) FROM "T3LVL2"."TABLE6" FOR READ ONLY
SELECT "Col86_DUP", "Col81_UNQ", "Col90_DUP", "Col89_DUP", "Col88_UNQ", "Col85_DUP", "Col83_UNQ", "C
SELECT 1 FROM "T3LVL2"."TABLE95" FOR UPDATE OF "Col1566_UNQ", "Col1570_UNQ", "Col1568_UNQ", "Col1567
SELECT COUNT(*) FROM "T3LVL2"."TABLE81" FOR READ ONLY
SELECT "Col89_DUP", "Col83_UNQ", "Col80_UNQ", "Col82_UNQ", "Col86_DUP", "Col88_UNQ", "Col81_UNQ" FRO
SELECT 1 FROM "T3LVL2"."TABLE92" FOR UPDATE OF "Col1518_DUP", "Col1519_UNQ", "Col1537_UNQ", "Col1536
SELECT "Col89_DUP", "Col83_UNQ", "Col80_UNQ", "Col82_UNQ", "Col86_DUP", "Col88_UNQ", "Col81_UNQ" FRO
SELECT "Col86_DUP", "Col81_UNQ", "Col90_DUP", "Col89_DUP", "Col88_UNQ", "Col85_DUP", "Col83_UNQ", "C
```

```
SELECT 1 FROM "T3LVL2"."TABLE61" FOR UPDATE OF "Col1022_DUP", "Col1035_UNQ", "Col1032_DUP", "Col1031
SELECT "Col352_DUP", "Col362_DUP", "Col350_UNQ", "Col351_UNQ", "Col357_DUP", "Col366_DUP", "Col356_U
SELECT 1 FROM "T3LVL2"."TABLE92" FOR UPDATE OF "Col1518_DUP", "Col1519_UNQ", "Col1537_UNQ", "Col1536
SELECT 1 FROM "T3LVL2"."TABLE10" FOR UPDATE OF "Col134_UNQ", "Col117_DUP", "Col105_DUP", "Col106_DUP
SELECT 1 FROM "T3LVL2"."TABLE65" FOR UPDATE OF "Col1088_DUP", "Col1078_UNQ", "Col1089_UNQ", "Col1087
SELECT "Col86_DUP", "Col81_UNQ", "Col90_DUP", "Col89_DUP", "Col88_UNQ", "Col85_DUP", "Col83_UNQ", "C
SELECT 1 FROM "T3LVL2"."TABLE23" FOR UPDATE OF "Col415_UNQ", "Col427_DUP", "Col410_UNQ", "Col419_UNQ
SELECT COUNT(*) FROM "T3LVL2"."TABLE6" FOR READ ONLY

  7521 record(s) selected.
```

Sample F2: Database summary of work characteristics by service subclass

This SQL query extracts key information about different characteristics of work that executed in each service subclass and summarizes it across all members. This information contains aggregate database metrics for resource consumption (summed across all members) as well as individual member high water marks for different characteristics (value shown is the highest value encountered at any member).

SQL text:

For an editable version of this SQL script, you can use the `sampleF2.sql` file included in the best practices `.zip` file.

```
--
-- Query to summarize basic characteristics of work executed in each
-- service class from a database perspective from the statistics event
-- monitor data
--
WITH

-- Determine when each member last started in order for delta metric calculations
  MEMBER_START_TIMES AS
     (SELECT DISTINCT SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP,
             MAX(MESSAGE_TIME) AS FIRSTCONNECT_TIME
      FROM CONTROL_DB2STATISTICS AS CONTROL, SCSTATS_DB2STATISTICS AS SCSTATS
      WHERE MESSAGE = 'FIRST_CONNECT'
         AND CONTROL.PARTITION_NUMBER = SCSTATS.PARTITION_NUMBER
         AND STATISTICS_TIMESTAMP > MESSAGE_TIME
      GROUP BY SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP),

-- Determine how to calculate deltas for each row
  DELTA_METHOD AS
     (SELECT SCSTATS.PARTITION_NUMBER,
             SCSTATS.STATISTICS_TIMESTAMP,
-- If previous row was gathered after last member start, determine delta
-- by subtraction
             CASE
               WHEN (LAG(SCSTATS.STATISTICS_TIMESTAMP, 1,
                       TIMESTAMP_FORMAT('2007-10-01 23:59:59',
                                        'YYYY-MM-DD HH24:MI:SS'))
                     OVER (PARTITION BY SCSTATS.PARTITION_NUMBER
                           ORDER BY SCSTATS.STATISTICS_TIMESTAMP))
                    >=
                    (SELECT FIRSTCONNECT_TIME
                     FROM MEMBER_START_TIMES AS REF
                     WHERE REF.PARTITION_NUMBER = SCSTATS.PARTITION_NUMBER
                        AND REF.STATISTICS_TIMESTAMP
                                        = SCSTATS.STATISTICS_TIMESTAMP)

               THEN 'Y'
               ELSE 'N'
             END AS SUBTRACT_FROM_PREV_ROW
      FROM SCSTATS_DB2STATISTICS AS SCSTATS
      GROUP BY SCSTATS.PARTITION_NUMBER, SCSTATS.STATISTICS_TIMESTAMP
      ORDER BY SCSTATS.PARTITION_NUMBER, SCSTATS.STATISTICS_TIMESTAMP),


-- extract wanted metrics from DETAILS_XML column in service class statistics
  V1 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             METRIC_NAME, VALUE
```

```
      FROM SCSTATS_DB2STATISTICS,
           TABLE (MON_FORMAT_XML_METRICS_BY_ROW(DETAILS_XML))
      WHERE METRIC_NAME IN ('TOTAL_CPU_TIME', 'ROWS_READ',
                            'ACT_RQSTS_TOTAL', 'ACT_COMPLETED_TOTAL')),


-- Pivot data into table format
  V2 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             MAX(DECODE(METRIC_NAME, 'ACT_RQSTS_TOTAL', VALUE))
                                                 AS ACT_RQSTS_TOTAL,
             MAX(DECODE(METRIC_NAME, 'ACT_COMPLETED_TOTAL', VALUE))
                                                 AS ACT_COMPLETED_TOTAL,
             MAX(DECODE(METRIC_NAME, 'TOTAL_CPU_TIME', VALUE))
                                                 AS TOTAL_CPU_TIME,
             MAX(DECODE(METRIC_NAME, 'ROWS_READ', VALUE)) AS TOTAL_ROWS_READ
      FROM V1
      GROUP BY STATISTICS_TIMESTAMP, PARTITION_NUMBER,
               SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME),


-- Calculate delta values for the extracted values based on delta calculation
-- method
  V3 AS
     (SELECT V2.STATISTICS_TIMESTAMP, V2.PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_RQSTS_TOTAL - LAG(ACT_RQSTS_TOTAL, 1, 0)
                                      OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                         SERVICE_SUPERCLASS_NAME,
                                                         SERVICE_SUBCLASS_NAME
                                            ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_RQSTS_TOTAL
             END AS ACT_RQSTS_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_COMPLETED_TOTAL - LAG(ACT_COMPLETED_TOTAL, 1, 0)
                                          OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                             SERVICE_SUPERCLASS_NAME,
                                                             SERVICE_SUBCLASS_NAME
                                                ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_COMPLETED_TOTAL
             END AS ACT_COMPLETED_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_CPU_TIME - LAG(TOTAL_CPU_TIME, 1, 0)
                                     OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                        SERVICE_SUPERCLASS_NAME,
                                                        SERVICE_SUBCLASS_NAME
                                           ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE TOTAL_CPU_TIME
             END AS TOTAL_CPU_TIME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_ROWS_READ - LAG(TOTAL_ROWS_READ, 1, 0)
                                      OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                         SERVICE_SUPERCLASS_NAME,
                                                         SERVICE_SUBCLASS_NAME
                                            ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE TOTAL_ROWS_READ
             END AS TOTAL_ROWS_READ
    FROM V2, DELTA_METHOD
    WHERE V2.PARTITION_NUMBER = DELTA_METHOD.PARTITION_NUMBER
       AND V2.STATISTICS_TIMESTAMP = DELTA_METHOD.STATISTICS_TIMESTAMP)


-- Summarize all metrics by day across all members
SELECT DATE(SOURCE.STATISTICS_TIMESTAMP) AS INTERVAL_DATE,
       SOURCE.SERVICE_SUPERCLASS_NAME,
       SOURCE.SERVICE_SUBCLASS_NAME,
       SUM(SOURCE.COORD_ACT_COMPLETED_TOTAL) AS COORD_REQ_COMPLETED,
       SUM(SOURCE.COORD_ACT_REJECTED_TOTAL) AS COORD_REQ_REJECTED,
       SUM(SOURCE.COORD_ACT_ABORTED_TOTAL) AS COORD_REQ_ABORTED,
       SUM(V3.ACT_COMPLETED_TOTAL) AS TOTAL_ACTIVITIES,
       SUM(V3.ACT_RQSTS_TOTAL) AS TOTAL_REQUESTS,
```

```
       SUM(V3.TOTAL_CPU_TIME) AS TOTAL_CPU_TIME,
       SUM(V3.TOTAL_ROWS_READ) AS TOTAL_ROWS_READ,
       MAX(SOURCE.ACT_ROWS_READ_TOP) AS ROWS_READ_TOP,
       MAX(SOURCE.ACT_CPU_TIME_TOP) AS CPU_TIME_TOP,
       MAX(SOURCE.CONCURRENT_ACT_TOP) AS MAX_CONCURRENT,
       MAX(SOURCE.UOW_TOTAL_TIME_TOP) AS UOW_TIME_TOP,
       MAX(SOURCE.ROWS_RETURNED_TOP) AS ROWS_RETURNED_TOP,
       MAX(SOURCE.COST_ESTIMATE_TOP) AS EST_COST_TOP,
       MAX(SOURCE.COORD_ACT_EST_COST_AVG) AS COORD_COST_AVG,
       MAX(SOURCE.COORD_ACT_LIFETIME_TOP) AS COORD_LIFE_TOP,
       MAX(SOURCE.COORD_ACT_LIFETIME_AVG) AS COORD_LIFE_AVG,
       MAX(SOURCE.COORD_ACT_EXEC_TIME_AVG) AS COORD_EXEC_AVG,
       MAX(SOURCE.COORD_ACT_QUEUE_TIME_AVG) AS COORD_QUEUE_AVG,
       MAX(SOURCE.COORD_ACT_INTERARRIVAL_TIME_AVG) AS COORD_INTERARRIVAL_AVG
FROM SCSTATS_DB2STATISTICS AS SOURCE, V3
WHERE SOURCE.STATISTICS_TIMESTAMP = V3.STATISTICS_TIMESTAMP
 AND SOURCE.PARTITION_NUMBER = V3.PARTITION_NUMBER
 AND SOURCE.SERVICE_SUPERCLASS_NAME = V3.SERVICE_SUPERCLASS_NAME
 AND SOURCE.SERVICE_SUBCLASS_NAME = V3.SERVICE_SUBCLASS_NAME
GROUP BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.SERVICE_SUPERCLASS_NAME,
         SOURCE.SERVICE_SUBCLASS_NAME
ORDER BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.SERVICE_SUPERCLASS_NAME,
         SOURCE.SERVICE_SUBCLASS_NAME;
```

Example output:

```
INTERVAL_DATE SERVICE_SUPERCLASS_NAME           SERVICE_SUBCLASS_NAME
------------- --------------------------------- -------------------------
10/20/2011    SYSDEFAULTMAINTENANCECLASS        SYSDEFAULTSUBCLASS
10/20/2011    SYSDEFAULTSYSTEMCLASS             SYSDEFAULTSUBCLASS
10/20/2011    SYSDEFAULTUSERCLASS               SYSDEFAULTSUBCLASS
10/20/2011    WLMBP_MASTER                      COMPLEX_DML
10/20/2011    WLMBP_MASTER                      ETL
10/20/2011    WLMBP_MASTER                      MEDIUM_DML
10/20/2011    WLMBP_MASTER                      MINOR_DML
10/20/2011    WLMBP_MASTER                      SIMPLE_DML
10/20/2011    WLMBP_MASTER                      SYSDEFAULTSUBCLASS
10/20/2011    WLMBP_MASTER                      TRIVIAL_DML

COORD_REQ_COMPLETED  COORD_REQ_REJECTED   COORD_REQ_ABORTED    TOTAL_ACTIVITIES
-------------------- -------------------- -------------------- --------------------
               41439                    0                  172                41439
                   0                    0                    0                    0
                 626                    0                    2                  626
                  10                    0                    0                    0
                   0                    0                    0                    0
                  67                    0                    0                    0
                1309                    0                   23                    0
                 254                    0                   29                    0
              120957                    0                  346              2109746
             1987982                    0                 1933                    0

TOTAL_REQUESTS       TOTAL_CPU_TIME       TOTAL_ROWS_READ      ROWS_READ_TOP
-------------------- -------------------- -------------------- --------------------
               68396            128179769              1126587                   -1
                   0               641443                  169                   -1
                 910             10911699                95259                   -1
                 810                17254                    8                    8
                   0                    0                    0                    0
                1962              1239892                29376                 1594
               22973              5666959               406598                20222
                6651              1430225                82065                 2098
              121451           8169555295               804156                   -1
             5573407            531475145             16079637              1575884

CPU_TIME_TOP         MAX_CONCURRENT UOW_TIME_TOP         ROWS_RETURNED_TOP
-------------------- -------------- -------------------- --------------------
                  -1              4                   -1                   -1
                  -1              0                   -1                   -1
                  -1              2                   -1                   -1
                1990              1                    0                    0
                   0              0                    0                    0
              805114              1                    0                  102
             1207285              7                    0                  477
```

```
              385339              8                   0                   99
                  -1             13                  -1                   -1
            24883934             60                   0                 1103

EST_COST_TOP         COORD_COST_AVG       COORD_LIFE_TOP       COORD_LIFE_AVG
-------------------- -------------------- -------------------- --------------------
                  -1                   -1                   -1                   -1
                  -1                   -1                   -1                   -1
                  -1                   -1                   -1                   -1
            45277628             45277628                 5823                 2208
                   0                    0                    0                    0
             2196364               450347               279884                24711
               29705                10277               405912                15877
              282657               231222               284552                81197
                  -1                   -1                   -1                   -1
                4993                   31               700813                  578

COORD_EXEC_AVG       COORD_QUEUE_AVG      COORD_INTERARRIVAL_AVG
-------------------- -------------------- ----------------------
                  -1                   -1                     -1
                  -1                   -1                     -1
                  -1                   -1                     -1
                2208                    0                1363084
                   0                    0                      0
                5817                    0                 257195
                5039                    0                  22074
                9436                    0                  87423
                  -1                   -1                     -1
                 390                    0                     15

  10 record(s) selected.
```

Sample F3: Database summary of work characteristics by workload

This SQL query extracts key information about different characteristics of work that was submitted for execution by each workload and summarizes it across all members. This information contains aggregate database metrics for resource consumption (summed across all members) as well as individual member high water marks for different characteristics (value shown is the highest value encountered at any member).

SQL text:

For an editable version of this SQL script, you can use the `sampleF3.sql` file included in the best practices `.zip` file.

```
--
-- Query to summarize basic characteristics of work executed from each
-- workload from a database perspective from the statistics event
-- monitor data
--
WITH

-- Determine when each member last started in order for delta metric calculations
  MEMBER_START_TIMES AS
     (SELECT DISTINCT WLSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP,
             MAX(MESSAGE_TIME) AS FIRSTCONNECT_TIME
      FROM CONTROL_DB2STATISTICS AS CONTROL, WLSTATS_DB2STATISTICS AS WLSTATS
      WHERE MESSAGE = 'FIRST_CONNECT'
         AND CONTROL.PARTITION_NUMBER = WLSTATS.PARTITION_NUMBER
         AND STATISTICS_TIMESTAMP > MESSAGE_TIME
      GROUP BY WLSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP),

-- Determine how to calculate deltas for each row
  DELTA_METHOD AS
     (SELECT WLSTATS.PARTITION_NUMBER,
             WLSTATS.STATISTICS_TIMESTAMP,
-- If previous row was gathered after last member start, determine delta
-- by subtraction
             CASE
               WHEN (LAG(WLSTATS.STATISTICS_TIMESTAMP, 1,
                         TIMESTAMP_FORMAT('2007-10-01 23:59:59',
```

```
                                    'YYYY-MM-DD HH24:MI:SS'))
                     OVER (PARTITION BY WLSTATS.PARTITION_NUMBER
                           ORDER BY WLSTATS.STATISTICS_TIMESTAMP))
                    >=
                    (SELECT FIRSTCONNECT_TIME
                     FROM MEMBER_START_TIMES AS REF
                     WHERE REF.PARTITION_NUMBER = WLSTATS.PARTITION_NUMBER
                        AND REF.STATISTICS_TIMESTAMP
                                       = WLSTATS.STATISTICS_TIMESTAMP)

              THEN 'Y'
              ELSE 'N'
            END AS SUBTRACT_FROM_PREV_ROW
     FROM WLSTATS_DB2STATISTICS AS WLSTATS
     GROUP BY WLSTATS.PARTITION_NUMBER, WLSTATS.STATISTICS_TIMESTAMP
     ORDER BY WLSTATS.PARTITION_NUMBER, WLSTATS.STATISTICS_TIMESTAMP),


-- extract wanted metrics from DETAILS_XML column in workload statistics
  V1 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             WORKLOAD_NAME,
             METRIC_NAME, VALUE
     FROM WLSTATS_DB2STATISTICS,
          TABLE (MON_FORMAT_XML_METRICS_BY_ROW(DETAILS_XML))
     WHERE METRIC_NAME IN ('TOTAL_CPU_TIME', 'ROWS_READ',
                           'ACT_RQSTS_TOTAL', 'ACT_COMPLETED_TOTAL')),


-- Pivot data into table format
  V2 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             WORKLOAD_NAME,
             MAX(DECODE(METRIC_NAME, 'ACT_RQSTS_TOTAL', VALUE))
                                                 AS ACT_RQSTS_TOTAL,
             MAX(DECODE(METRIC_NAME, 'ACT_COMPLETED_TOTAL', VALUE))
                                                 AS ACT_COMPLETED_TOTAL,
             MAX(DECODE(METRIC_NAME, 'TOTAL_CPU_TIME', VALUE))
                                                 AS TOTAL_CPU_TIME,
             MAX(DECODE(METRIC_NAME, 'ROWS_READ', VALUE)) AS TOTAL_ROWS_READ
     FROM V1
     GROUP BY STATISTICS_TIMESTAMP, PARTITION_NUMBER, WORKLOAD_NAME),


-- Calculate delta values for the extracted values based on delta calculation
-- method
  V3 AS
     (SELECT V2.STATISTICS_TIMESTAMP, V2.PARTITION_NUMBER,
             WORKLOAD_NAME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_RQSTS_TOTAL - LAG(ACT_RQSTS_TOTAL, 1, 0)
                                      OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                         WORKLOAD_NAME
                                            ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_RQSTS_TOTAL
             END AS ACT_RQSTS_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_COMPLETED_TOTAL - LAG(ACT_COMPLETED_TOTAL, 1, 0)
                                          OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                             WORKLOAD_NAME
                                                ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_COMPLETED_TOTAL
             END AS ACT_COMPLETED_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_CPU_TIME - LAG(TOTAL_CPU_TIME, 1, 0)
                                     OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                        WORKLOAD_NAME
                                           ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE TOTAL_CPU_TIME
             END AS TOTAL_CPU_TIME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_ROWS_READ - LAG(TOTAL_ROWS_READ, 1, 0)
                                      OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                         WORKLOAD_NAME
                                            ORDER BY V2.STATISTICS_TIMESTAMP)
```

```
            ELSE TOTAL_ROWS_READ
          END AS TOTAL_ROWS_READ
    FROM V2, DELTA_METHOD
    WHERE V2.PARTITION_NUMBER = DELTA_METHOD.PARTITION_NUMBER
       AND V2.STATISTICS_TIMESTAMP = DELTA_METHOD.STATISTICS_TIMESTAMP)


-- Summarize all metrics by day for each workload across all members
SELECT DATE(SOURCE.STATISTICS_TIMESTAMP) AS INTERVAL_DATE,
       SOURCE.WORKLOAD_NAME,
       SUM(SOURCE.COORD_ACT_COMPLETED_TOTAL) AS COORD_REQ_COMPLETED,
       SUM(SOURCE.COORD_ACT_REJECTED_TOTAL) AS COORD_REQ_REJECTED,
       SUM(SOURCE.COORD_ACT_ABORTED_TOTAL) AS COORD_REQ_ABORTED,
       SUM(V3.ACT_COMPLETED_TOTAL) AS TOTAL_ACTIVITIES,
       SUM(V3.ACT_RQSTS_TOTAL) AS TOTAL_REQUESTS,
       SUM(V3.TOTAL_CPU_TIME) AS TOTAL_CPU_TIME,
       SUM(V3.TOTAL_ROWS_READ) AS TOTAL_ROWS_READ,
       MAX(SOURCE.CONCURRENT_WLO_TOP) AS CONCURRENT_UOW_TOP,
       MAX(SOURCE.ACT_ROWS_READ_TOP) AS ROWS_READ_TOP,
       MAX(SOURCE.ACT_CPU_TIME_TOP) AS CPU_TIME_TOP,
       MAX(SOURCE.LOCK_WAIT_TIME_TOP) AS LOCKWAIT_TOP,
       MAX(SOURCE.UOW_TOTAL_TIME_TOP) AS UOW_TIME_TOP,
       MAX(SOURCE.ROWS_RETURNED_TOP) AS ROWS_RETURNED_TOP,
       MAX(SOURCE.COST_ESTIMATE_TOP) AS EST_COST_TOP,
       MAX(SOURCE.COORD_ACT_EST_COST_AVG) AS COORD_COST_AVG,
       MAX(SOURCE.COORD_ACT_LIFETIME_TOP) AS COORD_LIFE_TOP,
       MAX(SOURCE.COORD_ACT_LIFETIME_AVG) AS COORD_LIFE_AVG,
       MAX(SOURCE.COORD_ACT_EXEC_TIME_AVG) AS COORD_EXEC_AVG,
       MAX(SOURCE.COORD_ACT_QUEUE_TIME_AVG) AS COORD_QUEUE_AVG,
       MAX(SOURCE.COORD_ACT_INTERARRIVAL_TIME_AVG) AS COORD_INTERARRIVAL_AVG
FROM WLSTATS_DB2STATISTICS AS SOURCE, V3
WHERE SOURCE.STATISTICS_TIMESTAMP = V3.STATISTICS_TIMESTAMP
 AND SOURCE.PARTITION_NUMBER = V3.PARTITION_NUMBER
 AND SOURCE.WORKLOAD_NAME = V3.WORKLOAD_NAME
GROUP BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.WORKLOAD_NAME
ORDER BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.WORKLOAD_NAME;
```

Example output:

```
INTERVAL_DATE WORKLOAD_NAME            COORD_REQ_COMPLETED  COORD_REQ_REJECTED
------------- ---------------------- -------------------- --------------------
10/20/2011    SYSDEFAULTADMWORKLOAD                   208                    0
10/20/2011    SYSDEFAULTUSERWORKLOAD                2109675                    0

COORD_REQ_ABORTED    TOTAL_ACTIVITIES     TOTAL_REQUESTS       TOTAL_CPU_TIME
-------------------- -------------------- -------------------- --------------------
                   0                  208                  253              1761031
                2331              2110164              5726284           8717612715

TOTAL_ROWS_READ      CONCURRENT_UOW_TOP ROWS_READ_TOP        CPU_TIME_TOP
-------------------- ------------------ -------------------- --------------------
                2059                  2                   -1                   -1
            17488033                 65              1575884             24883934

LOCKWAIT_TOP         UOW_TIME_TOP         ROWS_RETURNED_TOP    EST_COST_TOP
-------------------- -------------------- -------------------- --------------------
                   0                   -1                   -1                   -1
           485035296             64257428                 1103             45277628

COORD_COST_AVG       COORD_LIFE_TOP       COORD_LIFE_AVG       COORD_EXEC_AVG
-------------------- -------------------- -------------------- --------------------
                  -1                   -1                   -1                   -1
                2236               700813                  531                  374

COORD_QUEUE_AVG      COORD_INTERARRIVAL_AVG
-------------------- ----------------------
                  -1                     -1
                   0                     13

  2 record(s) selected.
```

Sample F4: Member summary of work characteristics by service subclass

This SQL query extracts key information about different characteristics of work that executed in each service subclass and summarizes it per individual DB2 member. This information contains aggregate member metrics for resource consumption as well as individual high water marks for different characteristics (value shown is the highest value encountered at the member).

SQL text:

For an editable version of this SQL script, you can use the `sampleF4.sql` file included in the best practices `.zip` file.

```
--
-- Query to summarize basic characteristics of work executed in each
-- service class from an individual member perspective from the statistics event
-- monitor data
--
WITH


-- Determine when each member last started in order for delta metric calculations
  MEMBER_START_TIMES AS
     (SELECT DISTINCT SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP,
             MAX(MESSAGE_TIME) AS FIRSTCONNECT_TIME
      FROM CONTROL_DB2STATISTICS AS CONTROL, SCSTATS_DB2STATISTICS AS SCSTATS
      WHERE MESSAGE = 'FIRST_CONNECT'
         AND CONTROL.PARTITION_NUMBER = SCSTATS.PARTITION_NUMBER
         AND STATISTICS_TIMESTAMP > MESSAGE_TIME
      GROUP BY SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP),

-- Determine how to calculate deltas for each row
  DELTA_METHOD AS
     (SELECT SCSTATS.PARTITION_NUMBER,
             SCSTATS.STATISTICS_TIMESTAMP,
-- If previous row was gathered after last member start, determine delta
-- by subtraction
             CASE
               WHEN (LAG(SCSTATS.STATISTICS_TIMESTAMP, 1,
                         TIMESTAMP_FORMAT('2007-10-01 23:59:59',
                                          'YYYY-MM-DD HH24:MI:SS'))
                     OVER (PARTITION BY SCSTATS.PARTITION_NUMBER
                           ORDER BY SCSTATS.STATISTICS_TIMESTAMP))
                    >=
                    (SELECT FIRSTCONNECT_TIME
                     FROM MEMBER_START_TIMES AS REF
                     WHERE REF.PARTITION_NUMBER = SCSTATS.PARTITION_NUMBER
                        AND REF.STATISTICS_TIMESTAMP
                                          = SCSTATS.STATISTICS_TIMESTAMP)

               THEN 'Y'
               ELSE 'N'
             END AS SUBTRACT_FROM_PREV_ROW
      FROM SCSTATS_DB2STATISTICS AS SCSTATS
      GROUP BY SCSTATS.PARTITION_NUMBER, SCSTATS.STATISTICS_TIMESTAMP
      ORDER BY SCSTATS.PARTITION_NUMBER, SCSTATS.STATISTICS_TIMESTAMP),


-- extract wanted metrics from DETAILS_XML column in service class statistics
  V1 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             METRIC_NAME, VALUE
      FROM SCSTATS_DB2STATISTICS,
           TABLE (MON_FORMAT_XML_METRICS_BY_ROW(DETAILS_XML))
      WHERE METRIC_NAME IN ('TOTAL_CPU_TIME', 'ROWS_READ',
                            'ACT_RQSTS_TOTAL', 'ACT_COMPLETED_TOTAL')),


-- Pivot data into table format
  V2 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
```

```
              MAX(DECODE(METRIC_NAME, 'ACT_RQSTS_TOTAL', VALUE))
                                                    AS ACT_RQSTS_TOTAL,
              MAX(DECODE(METRIC_NAME, 'ACT_COMPLETED_TOTAL', VALUE))
                                                    AS ACT_COMPLETED_TOTAL,
              MAX(DECODE(METRIC_NAME, 'TOTAL_CPU_TIME', VALUE))
                                                    AS TOTAL_CPU_TIME,
              MAX(DECODE(METRIC_NAME, 'ROWS_READ', VALUE)) AS TOTAL_ROWS_READ
       FROM V1
       GROUP BY STATISTICS_TIMESTAMP, PARTITION_NUMBER,
                SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME),


-- Calculate delta values for the extracted values based on delta calculation
-- method
  V3 AS
     (SELECT V2.STATISTICS_TIMESTAMP, V2.PARTITION_NUMBER,
             SERVICE_SUPERCLASS_NAME,
             SERVICE_SUBCLASS_NAME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_RQSTS_TOTAL - LAG(ACT_RQSTS_TOTAL, 1, 0)
                                      OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                         SERVICE_SUPERCLASS_NAME,
                                                         SERVICE_SUBCLASS_NAME
                                            ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_RQSTS_TOTAL
             END AS ACT_RQSTS_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_COMPLETED_TOTAL - LAG(ACT_COMPLETED_TOTAL, 1, 0)
                                          OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                             SERVICE_SUPERCLASS_NAME,
                                                             SERVICE_SUBCLASS_NAME
                                                ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_COMPLETED_TOTAL
             END AS ACT_COMPLETED_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_CPU_TIME - LAG(TOTAL_CPU_TIME, 1, 0)
                                     OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                        SERVICE_SUPERCLASS_NAME,
                                                        SERVICE_SUBCLASS_NAME
                                           ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE TOTAL_CPU_TIME
             END AS TOTAL_CPU_TIME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_ROWS_READ - LAG(TOTAL_ROWS_READ, 1, 0)
                                      OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                         SERVICE_SUPERCLASS_NAME,
                                                         SERVICE_SUBCLASS_NAME
                                            ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE TOTAL_ROWS_READ
             END AS TOTAL_ROWS_READ
      FROM V2, DELTA_METHOD
      WHERE V2.PARTITION_NUMBER = DELTA_METHOD.PARTITION_NUMBER
         AND V2.STATISTICS_TIMESTAMP = DELTA_METHOD.STATISTICS_TIMESTAMP)

-- Summarize all metrics by day for each member
SELECT DATE(SOURCE.STATISTICS_TIMESTAMP) AS INTERVAL_DATE,
       SOURCE.PARTITION_NUMBER AS MEMBER_ID,
       SOURCE.SERVICE_SUPERCLASS_NAME,
       SOURCE.SERVICE_SUBCLASS_NAME,
       SUM(SOURCE.COORD_ACT_COMPLETED_TOTAL) AS COORD_REQ_COMPLETED,
       SUM(SOURCE.COORD_ACT_REJECTED_TOTAL) AS COORD_REQ_REJECTED,
       SUM(SOURCE.COORD_ACT_ABORTED_TOTAL) AS COORD_REQ_ABORTED,
       SUM(V3.ACT_COMPLETED_TOTAL) AS TOTAL_ACTIVITIES,
       SUM(V3.ACT_RQSTS_TOTAL) AS TOTAL_REQUESTS,
       SUM(V3.TOTAL_CPU_TIME) AS TOTAL_CPU_TIME,
       SUM(V3.TOTAL_ROWS_READ) AS TOTAL_ROWS_READ,
       MAX(SOURCE.ACT_ROWS_READ_TOP) AS ROWS_READ_TOP,
       MAX(SOURCE.ACT_CPU_TIME_TOP) AS CPU_TIME_TOP,
       MAX(SOURCE.CONCURRENT_ACT_TOP) AS MAX_CONCURRENT,
       MAX(SOURCE.UOW_TOTAL_TIME_TOP) AS UOW_TIME_TOP,
       MAX(SOURCE.ROWS_RETURNED_TOP) AS ROWS_RETURNED_TOP,
       MAX(SOURCE.COST_ESTIMATE_TOP) AS EST_COST_TOP,
       MAX(SOURCE.COORD_ACT_EST_COST_AVG) AS COORD_COST_AVG,
       MAX(SOURCE.COORD_ACT_LIFETIME_TOP) AS COORD_LIFE_TOP,
       MAX(SOURCE.COORD_ACT_LIFETIME_AVG) AS COORD_LIFE_AVG,
```

```
       MAX(SOURCE.COORD_ACT_EXEC_TIME_AVG) AS COORD_EXEC_AVG,
       MAX(SOURCE.COORD_ACT_QUEUE_TIME_AVG) AS COORD_QUEUE_AVG,
       MAX(SOURCE.COORD_ACT_INTERARRIVAL_TIME_AVG) AS COORD_INTERARRIVAL_AVG
FROM SCSTATS_DB2STATISTICS AS SOURCE, V3
WHERE SOURCE.STATISTICS_TIMESTAMP = V3.STATISTICS_TIMESTAMP
 AND SOURCE.PARTITION_NUMBER = V3.PARTITION_NUMBER
 AND SOURCE.SERVICE_SUPERCLASS_NAME = V3.SERVICE_SUPERCLASS_NAME
 AND SOURCE.SERVICE_SUBCLASS_NAME = V3.SERVICE_SUBCLASS_NAME
GROUP BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.PARTITION_NUMBER,
         SOURCE.SERVICE_SUPERCLASS_NAME,
         SOURCE.SERVICE_SUBCLASS_NAME
ORDER BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.PARTITION_NUMBER,
         SOURCE.SERVICE_SUPERCLASS_NAME,
         SOURCE.SERVICE_SUBCLASS_NAME;
```

Example output:

```
INTERVAL_DATE MEMBER_ID SERVICE_SUPERCLASS_NAME        SERVICE_SUBCLASS_NAME
------------- --------- ------------------------------ ----------------------
10/20/2011           10 SYSDEFAULTMAINTENANCECLASS     SYSDEFAULTSUBCLASS
10/20/2011           10 SYSDEFAULTSYSTEMCLASS          SYSDEFAULTSUBCLASS
10/20/2011           10 SYSDEFAULTUSERCLASS            SYSDEFAULTSUBCLASS
10/20/2011           10 WLMBP_MASTER                   COMPLEX_DML
10/20/2011           10 WLMBP_MASTER                   ETL
10/20/2011           10 WLMBP_MASTER                   MEDIUM_DML
10/20/2011           10 WLMBP_MASTER                   MINOR_DML
10/20/2011           10 WLMBP_MASTER                   SIMPLE_DML
10/20/2011           10 WLMBP_MASTER                   SYSDEFAULTSUBCLASS
10/20/2011           10 WLMBP_MASTER                   TRIVIAL_DML
10/20/2011           20 SYSDEFAULTMAINTENANCECLASS     SYSDEFAULTSUBCLASS
10/20/2011           20 SYSDEFAULTSYSTEMCLASS          SYSDEFAULTSUBCLASS
10/20/2011           20 SYSDEFAULTUSERCLASS            SYSDEFAULTSUBCLASS
10/20/2011           20 WLMBP_MASTER                   COMPLEX_DML
10/20/2011           20 WLMBP_MASTER                   ETL
10/20/2011           20 WLMBP_MASTER                   MEDIUM_DML
10/20/2011           20 WLMBP_MASTER                   MINOR_DML
10/20/2011           20 WLMBP_MASTER                   SIMPLE_DML
10/20/2011           20 WLMBP_MASTER                   SYSDEFAULTSUBCLASS
10/20/2011           20 WLMBP_MASTER                   TRIVIAL_DML
10/20/2011           30 SYSDEFAULTMAINTENANCECLASS     SYSDEFAULTSUBCLASS
10/20/2011           30 SYSDEFAULTSYSTEMCLASS          SYSDEFAULTSUBCLASS
10/20/2011           30 SYSDEFAULTUSERCLASS            SYSDEFAULTSUBCLASS
10/20/2011           30 WLMBP_MASTER                   COMPLEX_DML
10/20/2011           30 WLMBP_MASTER                   ETL
10/20/2011           30 WLMBP_MASTER                   MEDIUM_DML
10/20/2011           30 WLMBP_MASTER                   MINOR_DML
10/20/2011           30 WLMBP_MASTER                   SIMPLE_DML
10/20/2011           30 WLMBP_MASTER                   SYSDEFAULTSUBCLASS
10/20/2011           30 WLMBP_MASTER                   TRIVIAL_DML
10/20/2011           40 SYSDEFAULTMAINTENANCECLASS     SYSDEFAULTSUBCLASS
10/20/2011           40 SYSDEFAULTSYSTEMCLASS          SYSDEFAULTSUBCLASS
10/20/2011           40 SYSDEFAULTUSERCLASS            SYSDEFAULTSUBCLASS
10/20/2011           40 WLMBP_MASTER                   COMPLEX_DML
10/20/2011           40 WLMBP_MASTER                   ETL
10/20/2011           40 WLMBP_MASTER                   MEDIUM_DML
10/20/2011           40 WLMBP_MASTER                   MINOR_DML
10/20/2011           40 WLMBP_MASTER                   SIMPLE_DML
10/20/2011           40 WLMBP_MASTER                   SYSDEFAULTSUBCLASS
10/20/2011           40 WLMBP_MASTER                   TRIVIAL_DML

COORD_REQ_COMPLETED  COORD_REQ_REJECTED   COORD_REQ_ABORTED    TOTAL_ACTIVITIES
-------------------- -------------------- -------------------- --------------------
               41142                    0                  172                41142
                   0                    0                    0                    0
                 626                    0                    2                  626
                  10                    0                    0                    0
                   0                    0                    0                    0
                  67                    0                    0                    0
                1309                    0                   23                    0
                 254                    0                   29                    0
              120957                    0                  346              2109746
             1987982                    0                 1933                    0
                  99                    0                    0                   99
                   0                    0                    0                    0
                   0                    0                    0                    0
                   0                    0                    0                    0
```

```
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                   99                    0                    0                   99
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                   99                    0                    0                   99
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0
                    0                    0                    0                    0

TOTAL_REQUESTS       TOTAL_CPU_TIME       TOTAL_ROWS_READ      ROWS_READ_TOP
-------------------- -------------------- -------------------- --------------------
                68027            106704777              1126587                   -1
                    0               287693                  169                   -1
                  871              4847566                95181                   -1
                  210                17254                    0                    0
                    0                    0                    0                    0
                 1627               383977                 5893                 1594
                17197              3957955               154461                 7443
                 5134               692286                36146                  905
               121451           5887718297               738102                   -1
              3552603            519176059              5675310              1575884
                  123              7129145                    0                   -1
                    0               353750                    0                   -1
                   13              1271981                   26                   -1
                  200                    0                    0                    0
                    0                    0                    0                    0
                  111                 4589                 8779                 1461
                 1925               312137                49049                 5795
                  504               163820                10597                  424
                    0            747306013                19494                   -1
               669583              3381723              3081026                30173
                  123              7138197                    0                   -1
                    0                    0                    0                   -1
                   13              1370473                   26                   -1
                  200                    0                    0                    0
                    0                    0                    0                    0
                  111                 9592                 4450                  531
                 1925              1256513               112104                20222
                  504               120012                17201                 2098
                    0            763389488                22089                   -1
               673517              3215259              3537366                44199
                  123              7207650                    0                   -1
                    0                    0                    0                   -1
                   13              3421679                   26                   -1
                  200                    0                    8                    8
                    0                    0                    0                    0
                  113               841734                10254                 1061
                 1926               140354                90984                 9548
                  509               454107                18121                 1131
                    0            771141497                24471                   -1
               677704              5702104              3785935                81752

CPU_TIME_TOP         MAX_CONCURRENT UOW_TIME_TOP         ROWS_RETURNED_TOP
-------------------- -------------- -------------------- --------------------
                  -1              4                   -1                   -1
                  -1              0                   -1                   -1
                  -1              2                   -1                   -1
                1990              1                    0                    0
                   0              0                    0                    0
```

167050	1	0	102
1005594	7	0	477
169711	8	0	99
-1	13	-1	-1
24883934	60	0	1103
-1	2	-1	-1
-1	0	-1	-1
-1	0	-1	-1
0	0	0	0
0	0	0	0
2495	0	0	0
161517	0	0	0
127922	0	0	0
-1	0	-1	-1
779283	0	0	0
-1	2	-1	-1
-1	0	-1	-1
-1	0	-1	-1
0	0	0	0
0	0	0	0
3283	0	0	0
1207285	0	0	0
77636	0	0	0
-1	0	-1	-1
367183	0	0	0
-1	2	-1	-1
-1	0	-1	-1
-1	0	-1	-1
0	0	0	0
0	0	0	0
805114	0	0	0
23192	0	0	0
385339	0	0	0
-1	0	-1	-1
1046691	0	0	0

EST_COST_TOP	COORD_COST_AVG	COORD_LIFE_TOP	COORD_LIFE_AVG
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
45277628	45277628	5823	2208
0	0	0	0
2196364	450347	279884	24711
29705	10277	405912	15877
282657	231222	284552	81197
-1	-1	-1	-1
4993	31	700813	578
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
-1	-1	-1	-1
0	0	0	0
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
-1	-1	-1	-1
0	0	0	0
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
-1	-1	-1	-1
0	0	0	0

```
COORD_EXEC_AVG       COORD_QUEUE_AVG      COORD_INTERARRIVAL_AVG
-------------------- -------------------- ----------------------
                  -1                   -1                     -1
                  -1                   -1                     -1
                  -1                   -1                     -1
                2208                    0                1363084
                   0                    0                      0
                5817                    0                 257195
                5039                    0                  22074
                9436                    0                  87423
                  -1                   -1                     -1
                 390                    0                     15
                  -1                   -1                     -1
                  -1                   -1                     -1
                  -1                   -1                     -1
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                  -1                   -1                     -1
                   0                    0                      0
                  -1                   -1                     -1
                  -1                   -1                     -1
                  -1                   -1                     -1
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                  -1                   -1                     -1
                   0                    0                      0
                  -1                   -1                     -1
                  -1                   -1                     -1
                  -1                   -1                     -1
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                   0                    0                      0
                  -1                   -1                     -1
                   0                    0                      0

  40 record(s) selected.
```

Sample F5: Member summary of work characteristics by workload

This SQL query extracts key information about different characteristics of work that were submitted for execution by each workload and summarizes it per individual DB2 member. This information contains aggregate member metrics for resource consumption as well as individual high water marks for different characteristics (value shown is the highest value encountered at the member).

SQL text:

For an editable version of this SQL script, you can use the `sampleF5.sql` file included in the best practices `.zip` file.

```
--
-- Query to summarize basic characteristics of work executed from each
-- workload from an individual perspective from the statistics event
-- monitor data
--
WITH

-- Determine when each member last started in order for delta metric calculations
  MEMBER_START_TIMES AS
     (SELECT DISTINCT WLSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP,
             MAX(MESSAGE_TIME) AS FIRSTCONNECT_TIME
      FROM CONTROL_DB2STATISTICS AS CONTROL, WLSTATS_DB2STATISTICS AS WLSTATS
      WHERE MESSAGE = 'FIRST_CONNECT'
         AND CONTROL.PARTITION_NUMBER = WLSTATS.PARTITION_NUMBER
         AND STATISTICS_TIMESTAMP > MESSAGE_TIME
```

```
      GROUP BY WLSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP),

-- Determine how to calculate deltas for each row
  DELTA_METHOD AS
     (SELECT WLSTATS.PARTITION_NUMBER,
             WLSTATS.STATISTICS_TIMESTAMP,
-- If previous row was gathered after last member start, determine delta
-- by subtraction
             CASE
               WHEN (LAG(WLSTATS.STATISTICS_TIMESTAMP, 1,
                       TIMESTAMP_FORMAT('2007-10-01 23:59:59',
                                        'YYYY-MM-DD HH24:MI:SS'))
                     OVER (PARTITION BY WLSTATS.PARTITION_NUMBER
                           ORDER BY WLSTATS.STATISTICS_TIMESTAMP))
                    >=
                    (SELECT FIRSTCONNECT_TIME
                     FROM MEMBER_START_TIMES AS REF
                     WHERE REF.PARTITION_NUMBER = WLSTATS.PARTITION_NUMBER
                        AND REF.STATISTICS_TIMESTAMP
                                          = WLSTATS.STATISTICS_TIMESTAMP)

               THEN 'Y'
               ELSE 'N'
             END AS SUBTRACT_FROM_PREV_ROW
      FROM WLSTATS_DB2STATISTICS AS WLSTATS
      GROUP BY WLSTATS.PARTITION_NUMBER, WLSTATS.STATISTICS_TIMESTAMP
      ORDER BY WLSTATS.PARTITION_NUMBER, WLSTATS.STATISTICS_TIMESTAMP),


-- extract wanted metrics from DETAILS_XML column in workload statistics
  V1 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             WORKLOAD_NAME,
             METRIC_NAME, VALUE
      FROM WLSTATS_DB2STATISTICS,
           TABLE (MON_FORMAT_XML_METRICS_BY_ROW(DETAILS_XML))
      WHERE METRIC_NAME IN ('TOTAL_CPU_TIME', 'ROWS_READ',
                            'ACT_RQSTS_TOTAL', 'ACT_COMPLETED_TOTAL')),


-- Pivot data into table format
  V2 AS
     (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
             WORKLOAD_NAME,
             MAX(DECODE(METRIC_NAME, 'ACT_RQSTS_TOTAL', VALUE))
                                                  AS ACT_RQSTS_TOTAL,
             MAX(DECODE(METRIC_NAME, 'ACT_COMPLETED_TOTAL', VALUE))
                                                  AS ACT_COMPLETED_TOTAL,
             MAX(DECODE(METRIC_NAME, 'TOTAL_CPU_TIME', VALUE))
                                                  AS TOTAL_CPU_TIME,
             MAX(DECODE(METRIC_NAME, 'ROWS_READ', VALUE)) AS TOTAL_ROWS_READ
      FROM V1
      GROUP BY STATISTICS_TIMESTAMP, PARTITION_NUMBER, WORKLOAD_NAME),


-- Calculate delta values for the extracted values based on delta calculation
-- method
  V3 AS
     (SELECT V2.STATISTICS_TIMESTAMP, V2.PARTITION_NUMBER,
             WORKLOAD_NAME,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_RQSTS_TOTAL - LAG(ACT_RQSTS_TOTAL, 1, 0)
                                      OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                         WORKLOAD_NAME
                                            ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_RQSTS_TOTAL
             END AS ACT_RQSTS_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN ACT_COMPLETED_TOTAL - LAG(ACT_COMPLETED_TOTAL, 1, 0)
                                          OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                             WORKLOAD_NAME
                                                ORDER BY V2.STATISTICS_TIMESTAMP)
               ELSE ACT_COMPLETED_TOTAL
             END AS ACT_COMPLETED_TOTAL,
             CASE
               WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
               THEN TOTAL_CPU_TIME - LAG(TOTAL_CPU_TIME, 1, 0)
```

```
                                        OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                           WORKLOAD_NAME
                                              ORDER BY V2.STATISTICS_TIMESTAMP)
              ELSE TOTAL_CPU_TIME
            END AS TOTAL_CPU_TIME,
            CASE
              WHEN (SUBTRACT_FROM_PREV_ROW = 'Y')
              THEN TOTAL_ROWS_READ - LAG(TOTAL_ROWS_READ, 1, 0)
                                     OVER (PARTITION BY V2.PARTITION_NUMBER,
                                                        WORKLOAD_NAME
                                           ORDER BY V2.STATISTICS_TIMESTAMP)
              ELSE TOTAL_ROWS_READ
            END AS TOTAL_ROWS_READ
    FROM V2, DELTA_METHOD
    WHERE V2.PARTITION_NUMBER = DELTA_METHOD.PARTITION_NUMBER
       AND V2.STATISTICS_TIMESTAMP = DELTA_METHOD.STATISTICS_TIMESTAMP)

-- Summarize all metrics by day for each member
SELECT DATE(SOURCE.STATISTICS_TIMESTAMP) AS INTERVAL_DATE,
       SOURCE.PARTITION_NUMBER AS MEMBER_ID,
       SOURCE.WORKLOAD_NAME,
       SUM(SOURCE.COORD_ACT_COMPLETED_TOTAL) AS COORD_REQ_COMPLETED,
       SUM(SOURCE.COORD_ACT_REJECTED_TOTAL) AS COORD_REQ_REJECTED,
       SUM(SOURCE.COORD_ACT_ABORTED_TOTAL) AS COORD_REQ_ABORTED,
       SUM(V3.ACT_COMPLETED_TOTAL) AS TOTAL_ACTIVITIES,
       SUM(V3.ACT_RQSTS_TOTAL) AS TOTAL_REQUESTS,
       SUM(V3.TOTAL_CPU_TIME) AS TOTAL_CPU_TIME,
       SUM(V3.TOTAL_ROWS_READ) AS TOTAL_ROWS_READ,
       MAX(SOURCE.CONCURRENT_WLO_TOP) AS CONCURRENT_UOW_TOP,
       MAX(SOURCE.ACT_ROWS_READ_TOP) AS ROWS_READ_TOP,
       MAX(SOURCE.ACT_CPU_TIME_TOP) AS CPU_TIME_TOP,
       MAX(SOURCE.LOCK_WAIT_TIME_TOP) AS LOCKWAIT_TOP,
       MAX(SOURCE.UOW_TOTAL_TIME_TOP) AS UOW_TIME_TOP,
       MAX(SOURCE.ROWS_RETURNED_TOP) AS ROWS_RETURNED_TOP,
       MAX(SOURCE.COST_ESTIMATE_TOP) AS EST_COST_TOP,
       MAX(SOURCE.COORD_ACT_EST_COST_AVG) AS COORD_COST_AVG,
       MAX(SOURCE.COORD_ACT_LIFETIME_TOP) AS COORD_LIFE_TOP,
       MAX(SOURCE.COORD_ACT_LIFETIME_AVG) AS COORD_LIFE_AVG,
       MAX(SOURCE.COORD_ACT_EXEC_TIME_AVG) AS COORD_EXEC_AVG,
       MAX(SOURCE.COORD_ACT_QUEUE_TIME_AVG) AS COORD_QUEUE_AVG,
       MAX(SOURCE.COORD_ACT_INTERARRIVAL_TIME_AVG) AS COORD_INTERARRIVAL_AVG
FROM WLSTATS_DB2STATISTICS AS SOURCE, V3
WHERE SOURCE.STATISTICS_TIMESTAMP = V3.STATISTICS_TIMESTAMP
 AND SOURCE.PARTITION_NUMBER = V3.PARTITION_NUMBER
 AND SOURCE.WORKLOAD_NAME = V3.WORKLOAD_NAME
GROUP BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.PARTITION_NUMBER,
         SOURCE.WORKLOAD_NAME
ORDER BY DATE(SOURCE.STATISTICS_TIMESTAMP),
         SOURCE.PARTITION_NUMBER,
         SOURCE.WORKLOAD_NAME;
```

Example output:

```
INTERVAL_DATE MEMBER_ID WORKLOAD_NAME           COORD_REQ_COMPLETED  COORD_REQ_REJECTED
------------- --------- ----------------------- -------------------- --------------------
10/20/2011           10 SYSDEFAULTADMWORKLOAD                   208                    0
10/20/2011           10 SYSDEFAULTUSERWORKLOAD                2109675                    0
10/20/2011           20 SYSDEFAULTADMWORKLOAD                      0                    0
10/20/2011           20 SYSDEFAULTUSERWORKLOAD                     0                    0
10/20/2011           30 SYSDEFAULTADMWORKLOAD                      0                    0
10/20/2011           30 SYSDEFAULTUSERWORKLOAD                     0                    0
10/20/2011           40 SYSDEFAULTADMWORKLOAD                      0                    0
10/20/2011           40 SYSDEFAULTUSERWORKLOAD                     0                    0


COORD_REQ_ABORTED    TOTAL_ACTIVITIES     TOTAL_REQUESTS       TOTAL_CPU_TIME
-------------------- -------------------- -------------------- --------------------
                   0                  208                  253              1471331
                2331              2110164              3697592           6414969663
                   0                    0                    0                97679
                   0                    0               672215            752098184
                   0                    0                    0                97172
                   0                    0               676143            769092465
                   0                    0                    0                94849
                   0                    0               680334            781452403


TOTAL_ROWS_READ      CONCURRENT_UOW_TOP ROWS_READ_TOP        CPU_TIME_TOP
```

```
-------------------- ------------------ -------------------- --------------------
                2059                  2                   -1                   -1
             6701006                 65              1575884             24883934
                   0                  0                   -1                   -1
             3167504                  0                30173               779283
                   0                  0                   -1                   -1
             3691332                  0                44199              1207285
                   0                  0                   -1                   -1
             3928191                  0                81752              1046691

LOCKWAIT_TOP         UOW_TIME_TOP         ROWS_RETURNED_TOP    EST_COST_TOP
-------------------- -------------------- -------------------- --------------------
                   0                   -1                   -1                   -1
           480241242             64257428                 1103             45277628
                   0                   -1                   -1                   -1
           485035296                    0                    0                    0
                   0                   -1                   -1                   -1
           484681891                    0                    0                    0
                   0                   -1                   -1                   -1
           484575096                    0                    0                    0

COORD_COST_AVG       COORD_LIFE_TOP       COORD_LIFE_AVG       COORD_EXEC_AVG
-------------------- -------------------- -------------------- --------------------
                  -1                   -1                   -1                   -1
                2236               700813                  531                  374
                  -1                   -1                   -1                   -1
                   0                    0                    0                    0
                  -1                   -1                   -1                   -1
                   0                    0                    0                    0
                  -1                   -1                   -1                   -1
                   0                    0                    0                    0

COORD_QUEUE_AVG      COORD_INTERARRIVAL_AVG
-------------------- ----------------------
                  -1                     -1
                   0                     13
                  -1                     -1
                   0                      0
                  -1                     -1
                   0                      0
                  -1                     -1
                   0                      0

  8 record(s) selected.
```

Appendix G. Alternative approaches to statistical data analysis

This appendix is a compendium of useful topics on the subject of analyzing statistics event monitor data with an alternative approach.

Alternative approach to determining I/O impact

The **rows_read** metric provides a rough measure of the relative amount of access to the buffer pool or buffer pools, but it does not reflect the actual row size consumed and, therefore, the volume or amount of I/O performed. A more natural unit of measurement for buffer pool I/O is the page.

The following formulae can be used to measure I/O impact at the page level (for example, how many page accesses were made):

```
Pages read =
pool_temp_xda_l_reads + pool_temp_data_l_reads + pool_temp_index_l_reads +
               pool_xda_l_reads + pool_data_l_reads + pool_index_l_reads

Pages written =
pool_xda_writes + pool_data_writes + pool_index_writes

Direct I/O requests =
direct_writes + direct_reads
```

The component metrics used in these formulae are also available in the details_xml column and, if desired, can be extracted in the same manner as metrics such as **total_cpu_time** are extracted in the sample SQL statements that are provided.

Example SQL for post-processing of statistics event monitor data

As mentioned in the main body of this document, we do not recommend that you analyze the data in the statistics event monitor while it is being actively accessed by the event monitor because a potential conflict can occur. It is also desirable to supplement the raw information with indexes and the results of any common calculations to improve the performance of the analysis.

Although it is possible to simply shut down the statistics event monitor and add indexes before performing any analysis of the data, it is the usual goal of monitoring to be seamless and to provide a history over time so that trends and old behaviors can be explored. In such cases, a history of statistical information is kept and various forms of reports and analyses are run on that data at different points in time for different reasons. To this end, some example steps and SQL are provided here to outline the different aspects of managing the statistics event monitor data through its lifetime of usefulness.

There are many approaches to doing this work, such as using range partitioned tables to make management easier. The examples shown in this section are not intended as a best practice, by any means, and are provided solely for *inspirational* purposes as you consider your own need and use of historical statistics data.

The discussion that follows assumes that we have an implementation along the following lines:

- A defined monitoring table space across all administration nodes and all data nodes with sufficient space to hold statistical data collected over a full day

- Two statistics event monitors created with all tables present and using the table space mentioned in the preceding point
  - Only one of these statistics event monitors is defined as AUTOSTART to ensure that a statistics event monitor is always started when the database is activated. The other one is defined as MANUALSTART.
- The **wlm_collect_int** database configuration parameter is set to a value of 60 (collecting statistics every hour)
- For all workloads and service classes of significant interest, their definition has been set to COLLECT AGGREGATE ACTIVITY DATA BASE

With this environment outlined in the preceding list, we always have statistical information being gathered from all workload management entities every hour to the active statistics event monitor. In order to create and manage a history of this data, we need to gather it from the event monitor at regular intervals and place it into a set of historical tables. We do this by toggling the two event monitors so that the inactive one becomes active to take over the monitoring duties and the active one becomes inactive so that its data can be analyzed. If the database is brought down for some reason, then we are assured that the AUTOSTART statistics event monitor is present and collecting data.

The gathering of event monitor data, as described earlier, can be done using an approach similar to the following steps:

1. Create a `statistics_evmon_mgmt` script to do the following tasks:
   a. Determine which event monitor is active (that is, which is the primary event monitor)
   b. If there is data in the inactive secondary event monitor tables, then do the following steps. This scenario, where we have two event monitors with data, occurs when the MANUALSTART event monitor definition was actively collecting when the database was brought down because it would not be the one to automatically start when the database was restarted.
      1) Extract the data (see: “Extracting data”)
      2) Truncate all the tables for the secondary event monitor
      3) Commit
   c. Activate the inactive secondary event monitor
   d. Deactivate the active primary event monitor
   e. Extract the data from the primary event monitor (see: “Extracting data”)
   f. Truncate all the tables for the primary event monitor
   g. Commit
2. Define a `cron` job to run the `statistics_evmon_mgmt` script at some time after midnight, but before 1 AM which is the next scheduled collection of statistics by the DB2 database manager (for example, between 00:15 and 00:45)
3. Perform any post-processing on the extracted data, as required

Extracting data

The basic job of the extraction is to copy the data to a set of historical tables where it can eventually be analyzed. As such, the minimum that needs to be done is to simply copy over the raw data to a duplicate table.

For most analyses, we usually want to extract data from the DETAILS_XML column and calculate deltas from that data. We also want to store this extra data so that the work has to be done only once. When these other activities occur is for

you to decide, but we assume that the process to get the metrics out of the DETAILS_XML column happens during the extraction time while deltas are done during the post-processing phase.

The following DDL statement provides an idea of an initial historical table for the service class statistics from the DB2 for Linux, UNIX, and Windows Version 9.7 Fix Pack 4 statistics event monitor:

```
CREATE TABLE HISTORICAL_SCSTATS
   (
-- original SCSTATS columns
      PARTITION_KEY INTEGER NOT NULL,
      ACT_CPU_TIME_TOP BIGINT NOT NULL,
      ACT_REMAPPED_IN BIGINT NOT NULL,
      ACT_REMAPPED_OUT BIGINT NOT NULL,
      ACT_ROWS_READ_TOP BIGINT NOT NULL,
      AGG_TEMP_TABLESPACE_TOP BIGINT NOT NULL,
      CONCURRENT_ACT_TOP INTEGER NOT NULL,
      CONCURRENT_CONNECTION_TOP INTEGER NOT NULL,
      CONCURRENT_WLO_TOP INTEGER NOT NULL,
      COORD_ACT_ABORTED_TOTAL BIGINT NOT NULL,
      COORD_ACT_COMPLETED_TOTAL BIGINT NOT NULL,
      COORD_ACT_EST_COST_AVG BIGINT NOT NULL,
      COORD_ACT_EXEC_TIME_AVG BIGINT NOT NULL,
      COORD_ACT_INTERARRIVAL_TIME_AVG BIGINT NOT NULL,
      COORD_ACT_LIFETIME_AVG BIGINT NOT NULL,
      COORD_ACT_LIFETIME_TOP BIGINT NOT NULL,
      COORD_ACT_QUEUE_TIME_AVG BIGINT NOT NULL,
      COORD_ACT_REJECTED_TOTAL BIGINT NOT NULL,
      COST_ESTIMATE_TOP BIGINT NOT NULL,
      DETAILS_XML BLOB(1048576) LOGGED NOT COMPACT NOT NULL,
      LAST_WLM_RESET TIMESTAMP NOT NULL,
      PARTITION_NUMBER SMALLINT NOT NULL,
      REQUEST_EXEC_TIME_AVG BIGINT NOT NULL,
      ROWS_RETURNED_TOP BIGINT NOT NULL,
      SERVICE_CLASS_ID INTEGER NOT NULL,
      SERVICE_SUBCLASS_NAME VARCHAR(128) NOT NULL,
      SERVICE_SUPERCLASS_NAME VARCHAR(128) NOT NULL,
      STATISTICS_TIMESTAMP TIMESTAMP NOT NULL,
      TEMP_TABLESPACE_TOP BIGINT NOT NULL,
      UOW_TOTAL_TIME_TOP BIGINT NOT NULL,
-- control column
      ROW_PROCESSED CHAR(1),
-- extracted columns
      ACT_RQSTS_TOTAL BIGINT NOT NULL,
      ACT_COMPLETED_TOTAL BIGINT NOT NULL,
      ACT_ABORTED_TOTAL BIGINT NOT NULL,
      ACT_REJECTED_TOTAL BIGINT NOT NULL,
      TOTAL_CPU_TIME BIGINT NOT NULL,
      ROWS_READ BIGINT NOT NULL
   );
```

In the preceding example, the historical table contains all the original fields from the SCSTATS event monitor table, a set of columns for metrics extracted from the DETAILS_XML column, and a control column to be used later in post-processing. A similar table can be defined for the WLSTATS table, while the remaining WCSTATS, QSTATS, and HISTOGRAMBIN statistics event monitor tables do not need the control or extracted data columns because they do not contain such information at this time.

To populate the HISTORICAL_SCSTATS table, the following INSERT statement pulls data from the statistics event monitor and extracts the raw metrics from the DETAILS_XML column during that process. This latter step can also be done in the post-processing stage, if the time taken to do the extract was considered too long:

```
INSERT INTO HISTORICAL_SCSTATS
   WITH
      V1 AS (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
                    SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME,
                    METRIC_NAME, VALUE
             FROM SCSTATS_DB2STATISTICS,
                    TABLE(MON_FORMAT_XML_METRICS_BY_ROW(DETAILS_XML))
             WHERE METRIC_NAME IN ('ACT_RQSTS_TOTAL', 'ACT_COMPLETED_TOTAL',
                                   'ACT_ABORTED_TOTAL', 'ACT_REJECTED_TOTAL',
                                   'TOTAL_CPU_TIME', 'ROWS_READ')),

-- Pivot data into table format
   EXTRACTED AS (SELECT STATISTICS_TIMESTAMP, PARTITION_NUMBER,
                  SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME,
                  MAX(DECODE(METRIC_NAME,  'ACT_RQSTS_TOTAL', VALUE))
                             AS ACT_RQSTS_TOTAL,
                  MAX(DECODE(METRIC_NAME, 'ACT_COMPLETED_TOTAL', VALUE))
                             AS ACT_COMPLETED_TOTAL,
                  MAX(DECODE(METRIC_NAME, 'ACT_ABORTED_TOTAL', VALUE))
                             AS ACT_ABORTED_TOTAL,
                  MAX(DECODE(METRIC_NAME, 'ACT_REJECTED_TOTAL', VALUE))
                             AS ACT_REJECTED_TOTAL,
                  MAX(DECODE(METRIC_NAME, 'TOTAL_CPU_TIME', VALUE))
                             AS TOTAL_CPU_TIME,
                  MAX(DECODE(METRIC_NAME, 'ROWS_READ', VALUE)) AS ROWS_READ
           FROM V1
           GROUP BY STATISTICS_TIMESTAMP, PARTITION_NUMBER,
                    SERVICE_SUPERCLASS_NAME, SERVICE_SUBCLASS_NAME)

   SELECT SCSTATS.*,
                   'N',
                  EXTRACTED.ACT_RQSTS_TOTAL,
                  EXTRACTED.ACT_COMPLETED_TOTAL,
                  EXTRACTED.ACT_ABORTED_TOTAL,
                  EXTRACTED.ACT_REJECTED_TOTAL,
                  EXTRACTED.TOTAL_CPU_TIME,
                  EXTRACTED.ROWS_READ
    FROM SCSTATS_DB2STATISTICS AS SCSTATS,
                EXTRACTED
   WHERE SCSTATS.STATISTICS_TIMESTAMP = EXTRACTED.STATISTICS_TIMESTAMP
        AND SCSTATS.PARTITION_NUMBER = EXTRACTED.PARTITION_NUMBER
        AND SCSTATS.SERVICE_SUPERCLASS_NAME = EXTRACTED.SERVICE_SUPERCLASS_NAME
        AND SCSTATS.SERVICE_SUBCLASS_NAME = EXTRACTED.SERVICE_SUBCLASS_NAME;
```

A similar statement can be used to extract from the WLSTATS event monitor table while the other statistics event monitor tables can be extracted with simple INSERT over sub-select statements (for example, INSERT INTO HISTORICAL_WCSTATS SELECT WCSTATS.* FROM WCSTATS_DB2STATISTICS AS WCSTATS) because they do not contain a DETAILS_XML column with additional in-depth metrics.

The extract process must also capture one other set of information to allow us to perform proper delta calculations for the extracted metrics. The way that the delta value is calculated depends on whether the previous statistics row exists and is from the same contiguous database activation. This information is derived from data contained in the CONTROL table of the statistics event monitor in the form of FIRSTCONNECT messages. For all the data being extracted from the event monitor tables, we want to know the relationship of the statistics timestamp (when statistics were collected) and the nearest preceding FIRSTCONNECT message (when the database last came up before the collection). From this, we can decide the correct delta calculation method later on during post-processing of the extracted data.

To store this control information, we can create a simple historical table with the following SQL example:

```
CREATE TABLE HISTORICAL_TIMEDATA
  (
      PARTITION_NUMBER SMALLINT NOT NULL,
      STATISTICS_TIMESTAMP TIMESTAMP NOT NULL,
      FIRSTCONNECT_TIME TIMESTAMP NOT NULL
  );

CREATE UNIQUE INDEX TIMEDATA_INDEX1 ON HISTORICAL_TIMEDATA
   (
      PARTITION_NUMBER,
      STATISTICS_TIMESTAMP,
      FIRSTCONNECT_TIME
   );
```

We can then populate the historical table, with the data being extracted, using the following SQL statement example:

```
INSERT INTO HISTORICAL_TIMEDATA
  SELECT SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP,
                  MAX(MESSAGE_TIME) AS FIRSTCONNECT_TIME
  FROM CONTROL_DB2STATISTICS AS CONTROL, SCSTATS_DB2STATISTICS AS SCSTATS
  WHERE MESSAGE = 'FIRST_CONNECT'
        AND CONTROL.PARTITION_NUMBER = SCSTATS.PARTITION_NUMBER
        AND STATISTICS_TIMESTAMP > MESSAGE_TIME
GROUP BY SCSTATS.PARTITION_NUMBER, STATISTICS_TIMESTAMP;
```

This one table can be used by all the tables because the control table is shared across the event monitor and the same statistics timestamp is used for all rows in any table produced by the same statistics collection.

Post-processing data

This final stage is intended to prepare the data for future analysis by creating desired indexes, precalculating any common formulae, and producing the delta values for any metrics extracted from the DETAILS_XML column for the SCSTATS or WLSTATS tables. This processing can be done at any time after extraction and affects only the historical tables, not the actual statistics event monitor tables.

In our simple example, we need to calculate only the delta values for the extracted data. Let us assume that we choose to put the calculated deltas into a separate HISTORICAL_SCSTATS_DELTAS table from the raw, extracted data using the following SQL example:

```
CREATE TABLE HISTORICAL_SCSTATS_DELTAS
   (
      PARTITION_NUMBER SMALLINT NOT NULL,
      SERVICE_SUPERCLASS_NAME VARCHAR(128) NOT NULL,
      SERVICE_SUBCLASS_NAME VARCHAR(128) NOT NULL,
      STATISTICS_TIMESTAMP TIMESTAMP NOT NULL,
      DELTA_ACT_RQSTS_TOTAL BIGINT DEFAULT 0,
      DELTA_ACT_COMPLETED_TOTAL BIGINT DEFAULT 0,
      DELTA_ACT_ABORTED_TOTAL BIGINT DEFAULT 0,
      DELTA_ACT_REJECTED_TOTAL BIGINT DEFAULT 0,
      DELTA_TOTAL_CPU_TIME BIGINT DEFAULT 0,
      DELTA_ROWS_READ BIGINT DEFAULT 0
   );
```

To calculate the deltas, we take advantage of the ROW_PROCESSED control column to find the new rows in the HISTORICAL_SCSTATS table which need to have their delta values calculated from the metrics extracted from the DETAILS_XML column. You can use the following SQL example:

```
INSERT INTO HISTORICAL_SCSTATS_DELTAS
   WITH
     V1 AS (SELECT PARTITION_NUMBER, STATISTICS_TIMESTAMP, FIRSTCONNECT_TIME
                FROM HISTORICAL_TIMEDATA),

     V2 AS (SELECT SCSTATS.PARTITION_NUMBER,
                SERVICE_SUPERCLASS_NAME,
                SERVICE_SUBCLASS_NAME,
                SCSTATS.STATISTICS_TIMESTAMP,
                CASE
                 WHEN LAG(SCSTATS.STATISTICS_TIMESTAMP, 1,
                          TIMESTAMP_FORMAT('2007-10-01 23:59:59',
                                           'YYYY-MM-DD HH24:MI:SS'))
                          OVER (PARTITION BY SCSTATS.PARTITION_NUMBER,
                                SERVICE_SUPERCLASS_NAME,
                                SERVICE_SUBCLASS_NAME
                                ORDER BY SCSTATS.STATISTICS_TIMESTAMP)
                                >= FIRSTCONNECT_TIME
                 THEN 'Y'
                 ELSE 'N'
                END AS SUBTRACT_FROM_PREV_ROW
            FROM SCSTATS_DB2STATISTICS AS SCSTATS, V1
            WHERE SCSTATS.PARTITION_NUMBER = V1.PARTITION_NUMBER
             AND SCSTATS.STATISTICS_TIMESTAMP = V1.STATISTICS_TIMESTAMP
            ORDER BY PARTITION_NUMBER, SERVICE_SUPERCLASS_NAME,
                     SERVICE_SUBCLASS_NAME, STATISTICS_TIMESTAMP)

    SELECT SCSTATS.PARTITION_NUMBER,
                 SCSTATS.SERVICE_SUPERCLASS_NAME,
                 SCSTATS.SERVICE_SUBCLASS_NAME,
                 SCSTATS.STATISTICS_TIMESTAMP,
--
-- CALCULATE ACT_RQSTS_TOTAL DELTA
--
         CASE
-- WHEN PREVIOUS ROW WAS COLLECTED AFTER THE LAST MEMBER ACTIVATION
          WHEN (V2.SUBTRACT_FROM_PREV_ROW = 'Y')
-- THEN SUBTRACT FROM THE PREVIOUS ROW
          THEN ACT_RQSTS_TOTAL - LAG(ACT_RQSTS_TOTAL, 1, 0)
                                 OVER (PARTITION BY SCSTATS.PARTITION_NUMBER,
                                           SCSTATS.SERVICE_SUPERCLASS_NAME,
                                           SCSTATS.SERVICE_SUBCLASS_NAME
                                     ORDER BY SCSTATS.STATISTICS_TIMESTAMP)
-- ELSE RETURN ORIGINAL VALUE
          ELSE ACT_RQSTS_TOTAL
        END AS DELTA_ACT_RQSTS_TOTAL,
--
-- CALCULATE ACT_COMPLETED_TOTAL DELTA
--
         CASE
-- WHEN PREVIOUS ROW WAS COLLECTED AFTER THE LAST MEMBER ACTIVATION
          WHEN (V2.SUBTRACT_FROM_PREV_ROW = 'Y')
-- THEN SUBTRACT FROM THE PREVIOUS ROW
          THEN ACT_COMPLETED_TOTAL - LAG(ACT_COMPLETED_TOTAL, 1, 0)
                                     OVER (PARTITION BY SCSTATS.PARTITION_NUMBER,
                                               SCSTATS.SERVICE_SUPERCLASS_NAME,
                                               SCSTATS.SERVICE_SUBCLASS_NAME
                                         ORDER BY SCSTATS.STATISTICS_TIMESTAMP)
-- ELSE RETURN ORIGINAL VALUE
          ELSE ACT_COMPLETED_TOTAL
        END AS DELTA_ACT_COMPLETED_TOTAL,
--
-- CALCULATE ACT_ABORTED_TOTAL DELTA
--
         CASE
-- WHEN PREVIOUS ROW WAS COLLECTED AFTER THE LAST MEMBER ACTIVATION
          WHEN (V2.SUBTRACT_FROM_PREV_ROW = 'Y')
```

```
-- THEN SUBTRACT FROM THE PREVIOUS ROW
          THEN ACT_ABORTED_TOTAL - LAG(ACT_ABORTED_TOTAL, 1, 0)
                                   OVER (PARTITION BY SCSTATS.PARTITION_NUMBER,
                                                      SCSTATS.SERVICE_SUPERCLASS_NAME,
                                                      SCSTATS.SERVICE_SUBCLASS_NAME
                                         ORDER BY SCSTATS.STATISTICS_TIMESTAMP)
-- ELSE RETURN ORIGINAL VALUE
          ELSE ACT_ABORTED_TOTAL
        END AS DELTA_ACT_ABORTED_TOTAL,
--
-- CALCULATE ACT_REJECTED_TOTAL DELTA
--
         CASE
-- WHEN PREVIOUS ROW WAS COLLECTED AFTER THE LAST MEMBER ACTIVATION
          WHEN (V2.SUBTRACT_FROM_PREV_ROW = 'Y')
-- THEN SUBTRACT FROM THE PREVIOUS ROW
          THEN ACT_REJECTED_TOTAL - LAG(ACT_REJECTED_TOTAL, 1, 0)
                                    OVER (PARTITION BY SCSTATS.PARTITION_NUMBER,
                                                       SCSTATS.SERVICE_SUPERCLASS_NAME,
                                                       SCSTATS.SERVICE_SUBCLASS_NAME
                                          ORDER BY SCSTATS.STATISTICS_TIMESTAMP)
-- ELSE RETURN ORIGINAL VALUE
          ELSE ACT_REJECTED_TOTAL
        END AS DELTA_ACT_REJECTED_TOTAL,
--
-- CALCULATE TOTAL_CPU_TIME DELTA
--
         CASE
-- WHEN PREVIOUS ROW WAS COLLECTED AFTER THE LAST MEMBER ACTIVATION
          WHEN (V2.SUBTRACT_FROM_PREV_ROW = 'Y')
-- THEN SUBTRACT FROM THE PREVIOUS ROW
          THEN TOTAL_CPU_TIME - LAG(TOTAL_CPU_TIME, 1, 0)
                                OVER (PARTITION BY SCSTATS.PARTITION_NUMBER,
                                                   SCSTATS.SERVICE_SUPERCLASS_NAME,
                                                   SCSTATS.SERVICE_SUBCLASS_NAME
                                      ORDER BY SCSTATS.STATISTICS_TIMESTAMP)
-- ELSE RETURN ORIGINAL VALUE
          ELSE TOTAL_CPU_TIME
        END AS DELTA_TOTAL_CPU_TIME,
--
-- CALCULATE ROWS_READ DELTA
--
         CASE
-- WHEN PREVIOUS ROW WAS COLLECTED AFTER THE LAST MEMBER ACTIVATION
          WHEN (V2.SUBTRACT_FROM_PREV_ROW = 'Y')
-- THEN SUBTRACT FROM THE PREVIOUS ROW
          THEN ROWS_READ - LAG(ROWS_READ, 1, 0)
                           OVER (PARTITION BY SCSTATS.PARTITION_NUMBER,
                                              SCSTATS.SERVICE_SUPERCLASS_NAME,
                                              SCSTATS.SERVICE_SUBCLASS_NAME
                                 ORDER BY SCSTATS.STATISTICS_TIMESTAMP)
-- ELSE RETURN ORIGINAL VALUE
          ELSE ROWS_READ
        END AS DELTA_ROWS_READ
  FROM HISTORICAL_SCSTATS AS SCSTATS, V2
  WHERE SCSTATS.PARTITION_NUMBER = V2.PARTITION_NUMBER
        AND SCSTATS.SERVICE_SUPERCLASS_NAME = V2.SERVICE_SUPERCLASS_NAME
        AND SCSTATS.SERVICE_SUBCLASS_NAME = V2.SERVICE_SUBCLASS_NAME
        AND SCSTATS.STATISTICS_TIMESTAMP = V2.STATISTICS_TIMESTAMP
        AND SCSTATS.ROW_PROCESSED = 'N';
```

After this statement has successfully been executed so that all of the new rows have been processed, we then update all of the control column values in the HISTORICAL_SCSTATS table with a statement such as this:

```
UPDATE HISTORICAL_SCSTATS SET ROW_PROCESSED = 'Y' WHERE ROW_PROCESSED = 'N';
```

The final step in post-processing of the data involves the creation of indexes to help improve the performance of the queries used to analyze the statistical data. Although any number of indexes might be relevant (it depends on what analysis is being done), all of the historical tables benefit from an index similar to the following definition because they are common grouping identifiers:

```
CREATE INDEX SCSTATS_INDEX1 ON HISTORICAL_SCSTATS (and HISTORICAL_SCSTATS_DELTA)
  (
      PARTITION_NUMBER,
      SERVICE_SUPERCLASS_NAME,
      SERVICE_SUBCLASS_NAME,
      STATISTICS_TIMESTAMP
   );
```

A similar index, using the workload name instead of the service class name, can be useful on any data from the WLSTATS event monitor table.

Aggregating historical data

Depending on your needs, you might choose to indefinitely keep all of the individual rows generated per statistics collection and then delete the data when it is no longer needed. In other cases, although you might want to keep the data around for a longer period, you might not want or need such granularity, or have the storage available to keep it. In that case, you could choose to aggregate the individual collection data into rows representing the data for a coarser unit of time, such as a day, week, or month.

An example of such an approach might have us keeping the individual per hour collection data available for the last 2 weeks, keeping the 3rd and 4th most recent weeks of data aggregated from a per hour perspective up to a per day perspective. Older months can be further aggregated from a per day perspective up to a per week perspective, and then kept available for 6 months before discarding or archiving the data.

Index

T

W

Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Printed in USA